Graph Anomaly Detection for Unsupervised Attack Tracking

Lyes BOURENNANI
(supervisor: Pierre Parrend)

Technical Report n^o 1, July 2024
revision bf5e93f

Anomaly detection is already being used for unsupervised analysis to detect cyberattacks on real-world datasets. However, there is a missed opportunity to use communication information to improve attack detection. Therefore, visualizing real attack data as connected vertices and edges to represent communications and applying Graph Anomaly Detection (GAD) algorithms could be an approach to improve attack detection. There are several GAD algorithms, so having benchmarks to determine the best algorithm to choose for a given type of attack could be useful. These algorithms use hyperparameters and parameters, so identifying their contribution to attack detection is also important. The results would be added to the GPML library maintained by the Laboratoire de Recherche de l'EPITA, Security and Systems group, to provide easy access and integration of GAD algorithms.

Keywords
Graph Anomaly Detection, Structural anomalies, Real data benchmarking, SCAN [9], RADAR [5], ANOMALOUS [8], Attack detection, GPML

Laboratoire de Recherche de l'EPITA
14-16, rue Voltaire – FR-94276 Le Kremlin-Bicêtre CEDEX – France
Tél. +33 1 53 14 59 22 – Fax. +33 1 53 14 59 13
lyes.bourennani@epita.fr – http://www.lre.epita.fr/

Copying this document

Contents

1 Introduction **4**

2 Anomaly Detection **5**
- 2.1 Context of Anomaly Detection 5
- 2.2 Why use Graph Anomaly Detection ? 6
- 2.3 How to use Graph Anomaly Detection ? 6

3 State of the Art on Graph Anomaly Detection **7**
- 3.1 Types of anomalies 7
- 3.2 Interesting examples 8
- 3.3 Structural Clustering Algorithm for Networks [9] 9
- 3.4 Residual Analysis for Anomaly Detection in Attributed Networks [5] 10
- 3.5 A Joint Modeling Approach for Anomaly Detection on Attributed Networks [8] . 11

4 Evaluation of GAD for attack detection **13**
- 4.1 Unit Testing 13
- 4.2 Benchmarking on real data 13
  - 4.2.1 Real data benchmark without GAD 14
  - 4.2.2 Real data benchmark using SCAN [9] 15
  - 4.2.3 Real data benchmark using RADAR [5] 17
  - 4.2.4 Real data benchmark using ANOMALOUS [8] 18

5 Conclusion **20**
- 5.1 Conclusion 20
- 5.2 Future work 20

6 Bibliography **21**

Chapter 1

Introduction

This report explores the application of Graph Anomaly Detection (GAD) techniques to enhance the detection of cyberattacks by leveraging information in communications. The graph data structure is well adapted to visualize these communications. We could represent machines as vertices and the communications as edges, thus representing internet networks.

Current unsupervised anomaly detection methods primarily analyze individual data points or statistical properties. These methods often neglect the structural topology of communications. Our work focuses on filling this gap, by constructing graphs to represent communications and then applying GAD algorithms to enhance datasets with anomaly scores.

We provide a detailed evaluation of multiple GAD algorithms on real-world network communication data, identifying the most effective algorithms for different attack types. The implemented algorithms are integrated into the GPML library, providing an easy access and integration.

Chapter 2 discusses the context and importance of anomaly detection, traditional methods, and the motivation for using GAD. Chapter 3 reviews different types of anomalies, interesting examples, and the specific GAD algorithms studied. Chapter 4 gives details on the unit testing and real-data benchmarking of the selected GAD algorithms. Chapter 5 summarizes the results found and suggests future research directions.

Chapter 2

Anomaly Detection

2.1 Context of Anomaly Detection

Anomaly detection is a crucial aspect of data analysis, especially in fields where identifying unusual patterns can have significant implications, such as cyberattack detection. Anomaly detection is an unsupervised analysis of data that has no prior labeling. It aims to find anomalies in data by identifying patterns and deviations in the data's structure and statistical properties. The challenge, therefore, lies in the algorithm's ability to discern what constitutes a normal behavior and what constitutes an anomaly. We can cite two anomaly detection algorithms: Local Outlier Factor (LOF) [1] and Isolation Forest [6]. On the one hand, LOF [1] uses the concept of local density. It uses k-nearest neighbors to define the locality of data points and the distance between points gives the density, which determines outliers in data points. On the other hand, Isolation Forest [6] detects anomalies by isolating them. It relies on the fact that anomalies are few and different from normal data points.

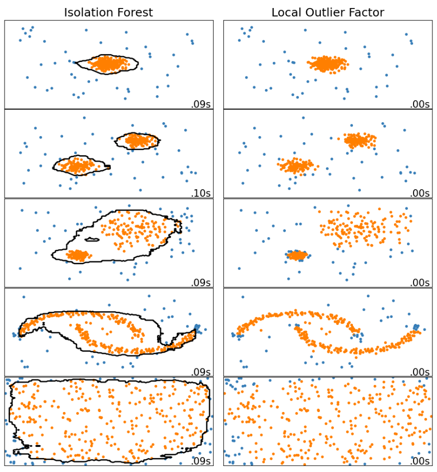

Figure 2.1: Comparison of Local Outlier Factor [1] and Isolation Forest [6] algorithms. [7]

2.2 Why use Graph Anomaly Detection ?

Statistical approaches are often inaccurate [4]. This is why machine learning approaches such as Cybersecurity Autonomous Machine Learning Platform for Anomaly Detection (CAMLPAD) [3] are used. However, these techniques are directly used on datasets. If these datasets represent the communications of network machines, for example, we still miss communications information. This is where Graph Anomaly Detection (GAD) could be used to represent communications, where vertices represent machines and edges represent communications between machines. Then, we could retrieve anomaly scores on machines using GAD algorithms to enhance attack detection.

2.3 How to use Graph Anomaly Detection ?

To use GAD for attack detection, we must identify the different characteristics of GAD algorithms, the different hyperparameters and parameters they use to determine how they contribute to information extraction. We also must establish the overall detection capabilities of GAD algorithms on attack types, since some algorithms could perform better on certain types of attacks. An approach to using GAD would be to benchmark the studied algorithms to determine the use cases.

Chapter 3

State of the Art on Graph Anomaly Detection

This part will describe the different types of anomalies, certain graph examples that help to understand the concepts of graph anomalies, and the GAD algorithms studied.

3.1 Types of anomalies

There are different types of graph anomalies. In an attributed network, we can have community anomalies, which are vertices that have outliers in the attributes of the corresponding community. A similar type of anomaly would be contextual anomalies, which are also vertices that have outliers but compared to all vertices of the network. There are also structural anomalies, which are determined by the structure of the networks. A vertex with only a few edges will have a higher anomaly score than a vertex that has many connections to a community. There are other types of anomalies, but these are the most notable for graph anomaly detection. This research focuses on structural anomalies to study the impact of network structures on attack detection.

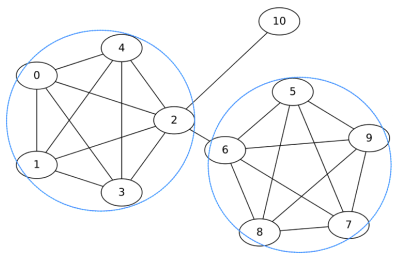

Figure 3.1: Graph example

Table 3.1: Attributes of the graph example

Vertex	Attribute 1	Attribute 2
0	3	11
1	2	9
2	12	8
3	4	12
4	3	13
5	32	9
6	35	7
7	32	24
8	33	12
9	31	9
10	34	10

On the graph example, we can see that vertex 2 is a community anomaly, since he has an Attribute 1 value too high for its community. Vertex 7 is a contextual anomaly, since its Attribute 2 value is too different from all the other vertices of the graph. We can also see that vertex 10 is a structural anomaly, since it possesses few connections to other vertices of the graph and does not belong to a community.

3.2 Interesting examples

There are examples of graphs that are interesting to see. For example, a complete graph is a good example because every vertex has the same structure, which means that in a non-attributed complete graph, every vertex would have the same anomaly score. In an attributed complete graph, only the attribute values would influence the anomaly scores. A complete graph could also represent the mesh topology for networking.

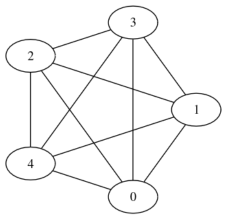

Figure 3.2: Complete Graph K5 Example with no attributes.

Another good example that was encountered during research was a star graph with two connected endpoints without attributes. When looking into the graph, we could say that vertices 7 and 9 are suspicious, but the anomaly scores will be smaller. This is due to the fact that these vertices possess more edges, and algorithms would believe that they are more connected.

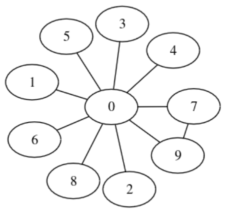

Figure 3.3: Star graph with two connected endpoints and no attributes.

3.3 Structural Clustering Algorithm for Networks [9]

The Structural Clustering Algorithm for Networks (SCAN) [9] is a clustering algorithm that labels each vertex of a graph into three categories: cluster member, hub or outlier. Hubs represent vertices that connect different clusters. Finally, outliers are the structural anomalies the algorithm finds. The algorithm is only designed for non-attributed networks. The algorithm has three main processing steps:

- Core identification: each vertex is checked to determine if it is a core vertex based on the neighborhood density.
- Cluster expansion: For core vertices, new clusters are expanded by finding all structure-reachable vertices.
- Classification of non-members: non-member vertices are further classified as hubs or outliers based on their connections to multiple clusters.

The algorithm uses two parameters named ϵ and μ. ϵ is a distance threshold parameter that defines the neighborhood of a vertex. μ is a threshold on the number of neighbors.

We define n the number of vertices and m the number of edges. The time complexity of SCAN [9] is linear with the number of edges m ($O(m)$), making it highly efficient for large-scale networks. The worst-case time complexity of the algorithm occurs when using it on complete graphs. The complexity is then quadratic with the number of vertices ($O(n^2)$).

Algorithm 1 SCAN Algorithm [9]

```
Require: Graph $G = (V, E)$, parameters $\epsilon$, $\mu$
Ensure: Clustering of vertices in $V$
 1: // all vertices in $V$ are labeled as unclassified;
 2: for each unclassified vertex $v \in V$ do
 3:     if CORE$_{\epsilon,\mu}(v)$ then
 4:         generate new clusterID;
 5:         insert all $x \in N_\epsilon(v)$ into queue $Q$;
 6:         while $Q \neq 0$ do
 7:             $y$ = first vertex in $Q$;
 8:             $R = \{x \in V \mid \text{DirREACH}_{\epsilon,\mu}(y, x)\}$;
 9:             for each $x \in R$ do
10:                 if $x$ is unclassified or non-member then
11:                     assign current clusterID to $x$;
12:                     if $x$ is unclassified then
13:                         insert $x$ into queue $Q$;
14:                     end if
15:                 end if
16:             end for
17:             remove $y$ from $Q$;
18:         end while
19:     else
20:         label $v$ as non-member;
21:     end if
22: end for
23: for each non-member vertex $v$ do
24:     if $\exists x, y \in \Gamma(v)$ such that $x$.clusterID $\neq$ $y$.clusterID then
25:         label $v$ as hub;
26:     else
27:         label $v$ as outlier;
28:     end if
29: end for
```

3.4 Residual Analysis for Anomaly Detection in Attributed Networks [5]

The Residual Analysis for Anomaly Detection in Attributed Networks (RADAR) [5] algorithm is designed for anomaly detection in attributed networks. It uses residual analysis to retrieve information from a given graph..

The RADAR [5] algorithm is characterized by the following:

- Residual Analysis: use residuals from attribute information and network structure to detect anomalies.
- Use of the Laplacian matrix: incorporate network topology via the Laplacian matrix.
- Iterative Optimization: update the residual matrix R and the matrix W to minimize the objective function.

The algorithm involves the following parameters:

- α: sparsity parameter for matrix W.
- β: number of anomalies. Also a sparsity parameter for the residual matrix R.
- γ: balances the residual analysis information. A high value will favor structural information and a low value will favor attribute information.

The RADAR [5] algorithm's time complexity is $i \times (O(n^2 d) + O(n^3))$, where d is the number of attributes for each node, n is the number of vertices in the graph and i is the number of optimization iterations. The time complexity is a limitation of the algorithm since it takes far more time than SCAN [9], for example.

Algorithm 2 RADAR Algorithm [5]

```
Require: Attribute matrix $X$, adjacency matrix $A$, parameters $\alpha, \beta, \gamma$
Ensure: Top $m$ instances with the highest anomaly scores
 1: Build Laplacian matrix $L$ from the adjacency matrix $A$
 2: Initialize $D_R$ and $D_W$ to be identity matrix
 3: Initialize $R = (I + \beta D_R + \gamma L)^{-1} X$
 4: while objective function does not converge do
 5:     Update $W$
 6:     Update $D_W$ by setting $D_W(i,i) = \frac{1}{2\|W(i,:)\|_2}$
 7:     Update $R$
 8:     Update $D_R$ by setting $D_R(i,i) = \frac{1}{2\|R(i,:)\|_2}$
 9: end while
10: Calculate the anomaly score for the $i$-th instance as $\|R(i,:)\|_2$
11: return Top $m$ instances with the highest anomaly scores
```

3.5 A Joint Modeling Approach for Anomaly Detection on Attributed Networks [8]

A Joint Modeling Approach for Anomaly Detection on Attributed Networks (ANOMALOUS) [8] algorithm is similar to RADAR [5]. It also leverages residual analysis to retrieve information from the graph. This implies that the algorithm also uses the Laplacian matrix of the graph and optimizes a residual matrix named $\tilde{R}$ and a matrix named W.

The main difference between ANOMALOUS [8] and RADAR [5] is the use of CUR decomposition:

$$\min_{C,U,R,\tilde{R}} \|X - CUR - \tilde{R}\|_F^2 + \Psi(\tilde{R}, \gamma, \varphi) \tag{3.1}$$

where C is a subset of columns, R is a subset of rows, U is the linking matrix, and $\tilde{R}$ is the residual matrix. The term Ψ controls the sparsity of $\tilde{R}$.

The ANOMALOUS [8] algorithm uses several parameters that control its behavior and influence the performance of anomaly detection. These parameters are described as follows:

- α: this parameter controls the sparsity of the instance selection.

- β: this parameter controls the sparsity of the attribute selection.
- γ: this parameter controls the sparsity of the residual matrix $\tilde{R}$.
- φ: it models the correlation between network structure and node attributes. A higher value of φ places more emphasis on structural information, and a smaller value places more emphasis on attribute information.

The ANOMALOUS [8] algorithm's time complexity is $i \times (O(n^2 d) + O(n^2))$, where d is the number of attributes for each node, n is the number of vertices in the graph and i is the number of optimization iterations.

Algorithm 3 ANOMALOUS [8] Algorithm

```
Require: Adjacency matrix $A$, attribute matrix $X$, parameters $\alpha, \beta, \gamma, \varphi$
Ensure: Top $t$ instances with the highest abnormal scores
 1: Build Laplacian matrix $L$ from the adjacency matrix $A$
 2: Initialize $D_R, D'_W$ and $D''_W$ to be identity matrices
 3: Initialize $\tilde{R} = X(I + \gamma D_R + \varphi L)^{-1}$
 4: Build orthogonal matrices $P, Q$ and diagonal matrices $\Theta_1, \Theta_2$
 5: while objective function in (6) not converged do
 6:     Update $W$
 7:     Update $D'_W$ by setting $D'_W(i,i) = \frac{1}{2\|W(i,:)\|_2}$
 8:     Update $D''_W$ by setting $D''_W(i,i) = \frac{1}{2\|W^T(i,:)\|_2}$
 9:     Update $\tilde{R}$
10:     Update $D_R$ by setting $D_R(i,i) = \frac{1}{2\|\tilde{R}^T(i,:)\|_2}$
11: end while
12: Compute the abnormal score for the $i$-th instance as $\|\tilde{R}(:,i)\|_2$ return Output top $t$ instances with the
    highest abnormal scores
```

Chapter 4

Evaluation of GAD for attack detection

The three presented algorithms were implemented in the GPML library. GPML is a library maintained by PhD students at the Laboratoire de Recherche de l'EPITA, in the Security and Systems group. GPML is a library of graph processing destined for machine learning. In order to test and evaluate the different algorithms, we divided the evaluation into two parts: Unit Testing and Evaluation on real data. The first part is used to verify if the implementations of algorithms are correct and determine the impacts of the different parameters. The second part is used to evaluate the contribution to attack detection.

4.1 Unit Testing

For unit testing, a set of graphs was created to evaluate if the algorithms were correctly implemented. The set also contributed to finding insights on network topologies. Using the set, creating unit tests for each algorithm was easier.

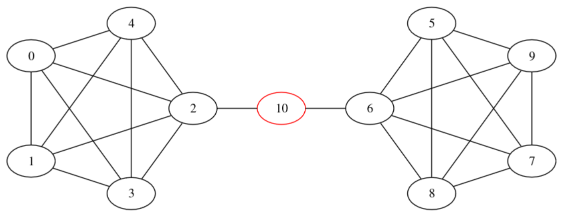

Figure 4.1: Graph extracted from the set. Two K5 connected by a vertex.

4.2 Benchmarking on real data

For real data benchmarking, we used the UGR16 dataset. It provides real attack data listed by spanish internet providers. The dataset contains tabular data of different packets emitted at a given time. Since it has a source and a destination IP address, we can build graphs and apply GAD algorithms. We used a sample of 50 000 entries, so we divided the sample into time window of five minutes to build time coherent graphs. Then, we applied GAD algorithms and retrieved anomaly scores to enhance the dataset. Finally, we ran classifiers to see if enhanced datasets had better results than the base dataset by comparing metrics (Precision, Recall, F1 Score and Balanced accuracy).

XGBoost offers better results than other classifiers since we are working with tabular data [2]. This is why we focused on XGBoost to determine if attack detection was enhanced.

Here are the different classifiers used:

- Decision Tree
- Random Forest
- Bagging with Decision Tree
- XGBoost

4.2.1 Real data benchmark without GAD

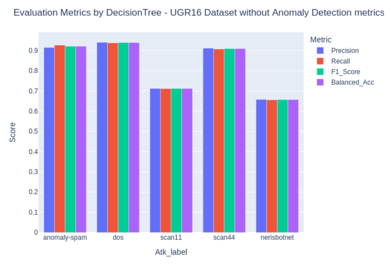

Figure 4.2: Decision Tree metrics without GAD

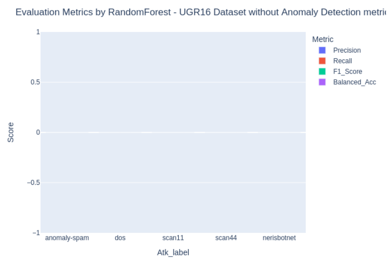

Figure 4.3: Random Forest metrics without GAD

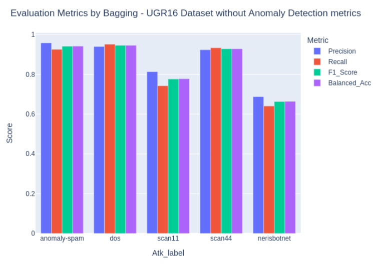

Figure 4.4: Bagging with Decision Tree metrics without GAD

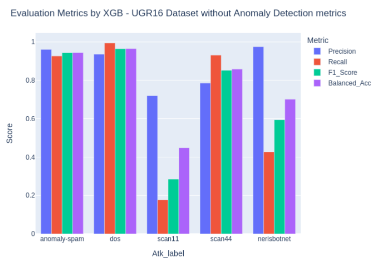

Figure 4.5: XGBoost metrics without GAD

4.2.2 Real data benchmark using SCAN [9]

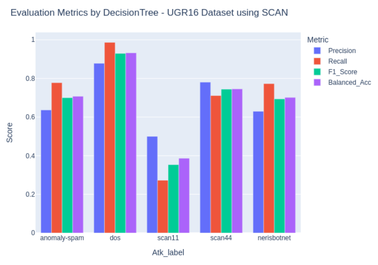

Figure 4.6: Decision Tree metrics using SCAN [9]

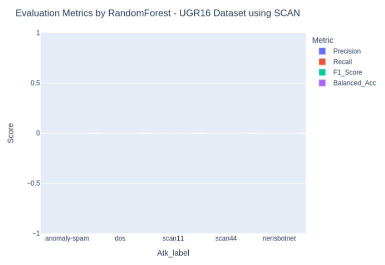

Figure 4.7: Random Forest metrics using SCAN [9]

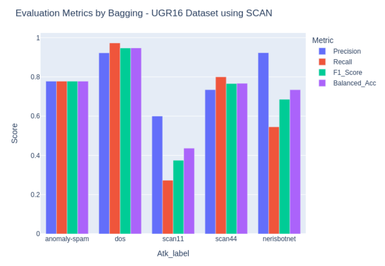

Figure 4.8: Bagging with Decision Tree metrics using SCAN [9]

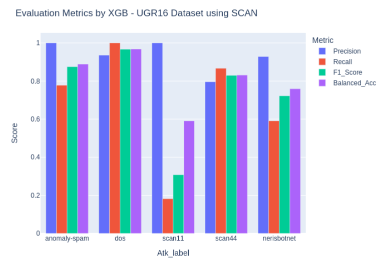

Figure 4.9: XGBoost metrics using SCAN [9]

If we compare XGBoost results, for the attack type ’SCAN11’, SCAN [9] algorithm adds 15% of balanced accuracy compared to the benchmark without GAD. We also see an increase of 5% for the ’NERISBOTNET’ attack type.

4.2.3 Real data benchmark using RADAR [5]

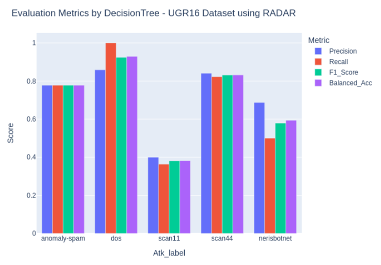

Figure 4.10: Decision Tree metrics using RADAR [5]

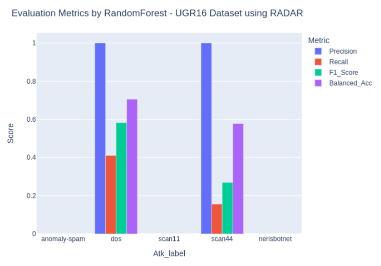

Figure 4.11: Random Forest metrics using RADAR [5]

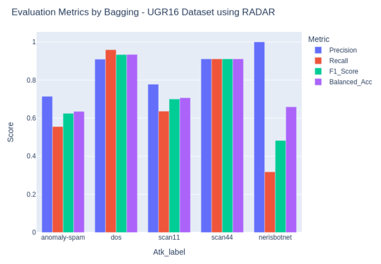

Figure 4.12: Bagging with Decision Tree metrics using RADAR [5]

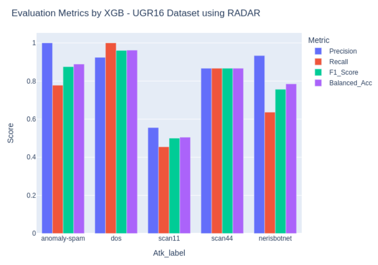

Figure 4.13: XGBoost metrics using RADAR [5]

We can observe an increase of 6% in the balanced accuracy of attack type ′SCAN11′ using RADAR [5] on the XGBoost classifier. There is also an increase of 8% in the attack type ′NERIS-BOTNET′. We have better metric results with Random Forest on ′DOS′ and ′SCAN44′.

4.2.4 Real data benchmark using ANOMALOUS [8]

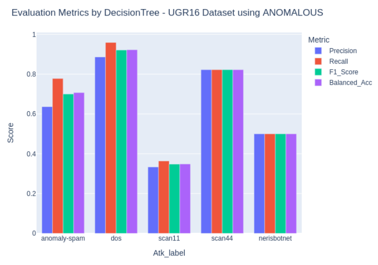

Figure 4.14: Decision Tree metrics using ANOMALOUS [8]

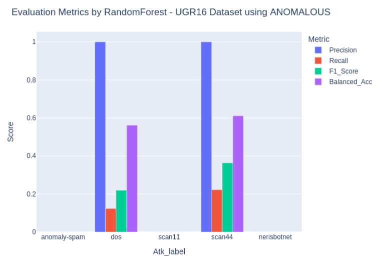

Figure 4.15: Random Forest metrics using ANOMALOUS [8]

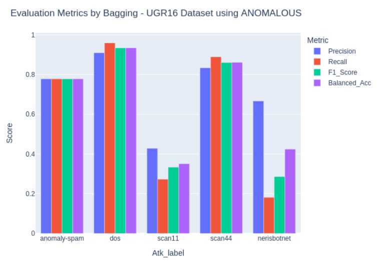

Figure 4.16: Bagging with Decision Tree metrics using ANOMALOUS [8]

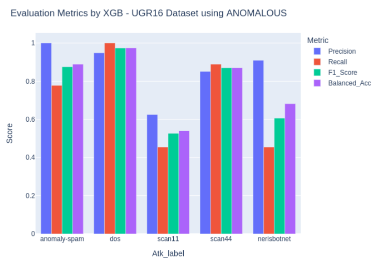

Figure 4.17: XGBoost metrics using ANOMALOUS [8]

Using ANOMALOUS [8], we only see an increase in the balanced accuracy of attack type 'SCAN11' of 9% in XGBoost. We also have better metric results with Random Forest on 'DOS' and 'SCAN44'. An interesting fact is that the results are quite similar to those of RADAR [5].

Chapter 5

Conclusion

5.1 Conclusion

In this report, we explored the application of Graph Anomaly Detection (GAD) techniques to enhance the detection of cyberattacks. We began by establishing the need for advanced anomaly detection methods that go beyond traditional approaches. Our investigation included a comprehensive review of the studied GAD algorithms, focusing on their ability to detect structural anomalies in network communication data.

The benchmarking revealed that GAD algorithms can enhance the detection of specific attack types, such as scanning and botnet attacks.

5.2 Future work

While we already have results, another iteration of benchmarking could reveal patterns in graph structure that could influence the anomaly scores. Another iteration of real data benchmarking to develop results on attack types could be useful. We also would like to benchmark the algorithms on their parameters to determine the best combinations of parameters.

Chapter 6

Bibliography

[1] Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104. (page 5)

[2] Grinsztajn, L., Oyallon, E., and Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520. (page 14)

[3] Hariharan, A., Gupta, A., and Pal, T. (2020). Camlpad: Cybersecurity autonomous machine learning platform for anomaly detection. In *Advances in Information and Communication: Proceedings of the 2020 Future of Information and Communication Conference (FICC), Volume 2*, pages 705–720. Springer. (page 6)

[4] Jaber, M., Boutry, N., and Parrend, P. (2024). Graph-based spectral analysis for detecting cyber attacks. In *ARES 2024 (The International Conference on Availability, Reliability and Security), Strasbourg, France, mai 2024*. (page 6)

[5] Li, J., Dani, H., Hu, X., and Liu, H. (2017). Radar: Residual analysis for anomaly detection in attributed networks. In *IJCAI*, volume 17, pages 2152–2158. (pages 1, 3, 10, 11, 17, 18, and 19)

[6] Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE. (page 5)

[7] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830. (page 5)

[8] Peng, Z., Luo, M., Li, J., Liu, H., Zheng, Q., et al. (2018). Anomalous: A joint modeling approach for anomaly detection on attributed networks. In *IJCAI*, pages 3513–3519. (pages 1, 3, 11, 12, 18, and 19)

[9] Xu, X., Yuruk, N., Feng, Z., and Schweiger, T. A. (2007). Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 824–833. (pages 1, 3, 9, 10, 11, 15, and 16)