pwc

UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering,
Mathematics & Computer Science

Automatic Detection of Misconfigurations of AWS Identity and Access Management Policies

Niek Khasuntsev
Master Thesis
25 June 2021

Graduation committee:
Dr.Ir. A. Continella
Dr. A. Sperotto
B. Steen, MSc

Services and CyberSecurity group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Automatic Detection of Misconfigurations of AWS Identity and Access Management Policies

Niek Khasuntsev
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
n.a.khasuntsev@student.utwente.nl

Abstract

Security misconfigurations are one of the biggest threats to cloud environments. In recent years, misconfigurations of cloud services have led to major security incidents and large-scale data breaches. Proper configuration of identity and access management services is essential in maintaining a secure cloud environment. Due to the dynamic and complex nature of cloud environments, misconfigurations can be easily introduced and go undetected for a long period. Therefore, it is critical to detect any potential misconfigurations before they can be abused.

In this paper, we present a novel misconfiguration detection approach for identity and access management policies in AWS. Our approach is based on a graph model representation of identity and access management data. We assume that similar identity and access management policies also have similar graph representations. Therefore, properly configured policies are similar to each other, and misconfigurations are different. Our main insight therefore is that we can use anomaly detection techniques to spot outliers, and therefore detect potential misconfigurations. Our proposed approach first creates a graph model from all the identity and access management policies in a cloud environment. Then, the graph is transformed into a vector representation. Finally, we apply anomaly detection on new observations to determine whether they are potential misconfigurations or not. We evaluate our approach on real-world identity and access management policy data of three cloud environments and demonstrate its effectiveness to detect misconfigurations (precision of 85%, recall of 73%).

1. INTRODUCTION

Data breaches are still an increasing threat to society [1]. A few years ago a data breach that compromised a hundred data records would have been major news. Nowadays, data breaches of hundreds of millions or even billions of data records can be considered common. In 2019, Capital One, an American bank holding suffered a data breach, where data of over a hundred of million people was stolen [2]. More recently, the credit card details of more than a hundred million hotel guests were stolen [3], the personal data of over 10 million churchgoers was leaked [4], and inmate records were leaked from a prison system [5]. These data breaches only demonstrate part of a much larger problem, as there are online lists that collect vulnerable cloud systems [6] [7]. All these breaches could have been prevented and have two characteristics in common. First, all the breached databases were hosted in the cloud. Second, the systems were vulnerable due to security misconfigurations in the used cloud resources, e.g. Amazon Simple Storage Service (S3) [8].

Ever since its introduction in 2006, cloud computing has been on the rise. Cloud computing allows for the delivery of on-demand computing services, which range from computing power to storage services. Due to the pay-as-you-go nature of the services, users can benefit from a flexible cost structure, with no considerable investments needed upfront. While the use of cloud services offers a wide range of benefits, it also comes with several security challenges [9]. As the aforementioned data breaches already demonstrate, security misconfigurations, like for example publicly accessible private data, are a considerable threat to cloud security. The problem is even larger than we see since only misconfigurations that lead to incidents are reported [10]. This is also confirmed in multiple surveys on cloud security [11] [12], where security misconfigurations are always listed as one of the most important security threats to cloud computing. In fact misconfigurations have a higher impact on cloud environments since these environments generally are exposed to the Internet, which creates a larger attack surface and makes it easier for malicious actors to abuse the misconfigurations.

To prevent the aforementioned breaches, proper identity and access management (IAM) is needed. IAM is about defining and managing the roles and access privileges of network users and systems. When configured

correctly, IAM systems prevent unauthorized access to the protected resources, ultimately making sure that only the right users get access to the right resources. However, errors can be made during the configuration of IAM systems: this is what we call misconfigurations. Examples of IAM misconfigurations can be:

- Attaching IAM policies to users instead of groups or roles;
- Granting all privileges to all users instead of adopting the least privilege principle (over permissive policies);
- Resource-based policies are not attached to the defined resource.

Cloud providers have adopted the shared responsibility model [13]. In essence, this means that cloud providers are responsible for the security of the cloud infrastructure and physical machines, while customers are responsible for the security of whatever is hosted in the cloud. Therefore, in this model, the configuration of identity and access management systems is the customer's responsibility [14]. However, cloud environments are often large, dynamic, and complex, which can make the configuration of security services difficult, intensive, and error-prone. Therefore, there is a need for systems to detect potential misconfigurations, before any misconfigurations can be abused.

Cloud Custodian [15] is a widely used open-source tool, which utilizes a rule-based approach to prevent the introduction of security misconfigurations in cloud environments. Each new resource is first compared against the created rules before it is deployed. Rule-based approaches are limited by the fact that rules need to be created and maintained to adhere to security policies. This can be error-prone and require a large effort. Furthermore, resources are only checked on creation, while in dynamic cloud environments misconfigurations can be introduced through changes to the policies or resources. A second widely used tool is P-DIFF [16], which is a tool for monitoring access and control behavior by using decision tree algorithms. P-DIFF relies on a reactive approach, and misconfigurations will be detected when they are already abused, which is also the tool's biggest limitation. Furthermore, P-DIFF learns access control policies from access logs and therefore it is limited to the information contained in the access logs.

To overcome the limitations of existing solutions, we propose a novel misconfiguration detection approach. We aim to detect potential misconfigurations in a fully automated, proactive, and generic way while requiring low effort and maintenance. First, we collect all identity and access management policies from cloud environments. Due to the connected nature of identity and access management policies, our approach relies on a graph-based model of the policies. Thus, we transform the collected policies into graph representations. We assume that similar policies also have similar graph representations. Properly configured policies are similar to each other and therefore, have similar graph representations. Misconfigurations, on the other hand, are generally over-permissive and therefore, are different. Based on this assumption, our main insight is that we can use anomaly detection techniques to detect potential misconfigurations.

For the implementation of our approach, we use Python for the policy retrieval and anomaly detection steps. For the graph model implementation, we use Neo4j [17], a graph database platform.

To validate our proposed approach, we have collected real-world identity and access management policy data of three AWS cloud environments from three different companies. We manually labeled the data for benign policies and potential misconfigurations, and then evaluated our proposed approach. On average our proposed approach has a precision of 85% and a recall of 73%.

In short, our paper makes the following contributions:

- We introduce an approach to model identity and access management policies, and the attached entities, using a graph-based representation;
- We present a novel misconfiguration detection system that, based on our approach, uses anomaly detection techniques to identify potential misconfigurations;
- We evaluate our proposed approach on real-world identity and access management policy data from three AWS cloud environments.

In spirit of open science, our Python source code implementing our approach is available at `https://github.com/utwente-scs/iam-collector`.

2. BACKGROUND AND MOTIVATION

In this section, we introduce the identity and access management policies used in AWS environments, which is the focus of this research project. Then, we discuss the current challenges in this domain and the goals of our proposed approach.

2.1 IAM in AWS

Identity and access management enables customers to manage their access to AWS services and resources securely. Using IAM, the customer can create and manage AWS users, groups, and roles, which are called entities. Furthermore, permissions can be used to allow and deny certain actions on AWS resources. Within AWS, IAM is offered as a dedicated service [18] which allows for fine-grained access control for all the resources.

```
{
   "Version":"2012-10-17",
   "name":"AdministratorAccess",
   "Statement":{
      "Effect":"Allow",
      "Action":"*",
      "Resource":"*"
   }
}
```

Figure 1: AdministratorAccess policy, allowing all actions on all resources

The permissions, which control the actions on resources, are captured in policies and are stored in JSON format. These policies can contain some top-level elements, like name and creation data, as well as one or multiple statements. Within statements, actions are specified, as well as the resources on which the actions work. Actions are either allowed or denied. An example of a policy can be found in Figure 1. First, we have some top-level elements, in this case, version and name, which are followed by a statement. The "*" character in AWS policies is a wildcard symbol and is equivalent to "all". Therefore, the above policy allows for all actions to be performed on all resources.

Before policies can be used, they need to be attached to an entity. This means either to a user, group, or role. The entity is then granted the permissions listed in the policy. In AWS, roles can be assumed by either users or other systems. By assuming a role, the user is then granted all the permissions listed in the policies attached to that role.

2.2 Problem Statement

Under the previously mentioned shared responsibility model [13] customers are responsible for the proper configuration of the IAM service in AWS. This means that the customers are responsible for the proper creation and maintenance of the identity and access management policies, as well as attaching the policies to the right entities. This process can become quite complex and demanding in cloud environments. Due to the dynamic nature of cloud environments, changes are made frequently, which means that misconfigurations can be introduced at any point in time, and can go undetected for a long period. Creating over permissive policies, or attaching policies to the wrong entities can lead to dangerous misconfigurations. As a consequence, misconfigurations often lead to large-scale data breaches and other high-impact security incidents. Therefore, a proper system is needed to detect potential misconfigurations before they can lead to security incidents.

2.3 Goals and Challenges

Detecting misconfigurations in cloud environments is a fairly new field of research, therefore there are only a limited number of existing solutions. We already discussed a few, the other ones we discuss in Section 8. Each existing solution has several limitations, rule-based approaches require a large effort to create and maintain rules, while other tools rely on a reactive approach after the misconfiguration is already abused. Therefore, there is a need for a novel misconfiguration detection approach. The new approach should meet the following requirements:

- **Fully automated**; The proposed solution should not rely on manually created rules and detect misconfigurations in a fully automated way;
- **Proactive**; Misconfigurations should be detected before they can be abused;
- **Generic**; The system should detect misconfigurations in a generic way, i.e. not only being able to detect a certain type of misconfiguration, but rather detect patterns of correctly configured resources such that all possible misconfigurations can be detected;
- **Low effort and maintenance**; Ideally the solution should be deployed and forgotten, requiring low effort and maintenance to keep the system functioning.

3. APPROACH

We aim to detect potential misconfigurations in identity and access management policies on the basis of the policy documents as well as the attached entities in the environment. Identity and access management policies by default have a connected nature. Policies connect actions to resources, as well as permissions to users. Therefore, our approach relies on a graph-based representation of the policies. The graph representation enables us to leverage the connected nature of policies, and as an added benefit provides advantages for the analysis and visualization of the policies.

We assume that when policies are similar to each other, it also results in similar graph representations. The similarity between policies is defined by the level of permissiveness, meaning the specified amount of allowed actions and resources, type of allowed actions (e.g. read-only), as well as the amount and type of attached entities. Based on this assumption, we also assume that properly configured policies have similar levels of permissiveness, and therefore have similar graph representations. Misconfigurations, on the other hand, are generally far more permissive, and therefore are different compared to properly configured policies, which makes misconfigurations stand out.

Furthermore, due to the dynamic nature of cloud environments, misconfigurations can be introduced at various points in time, through various changes to the policies. Modifications can be made to existing policies, new policies can be created, or policies can be removed. Therefore, new observations must also be considered in our approach.

Based on these assumptions and observations, our main insight is that anomaly detection techniques [19] capture misconfigurations, and therefore enable the detection of misconfigurations. To achieve this, the anomaly detection models need to be trained on properly configured, or benign, policy data. Then, when new observations occur, they need to be evaluated and checked.

In summary, to detect misconfigurations in identity and access management policies, we make the following design decisions:

- The methodology uses a graph-based approach to model IAM policies and their attached entities in graph representations;
- The methodology is based on anomaly detection techniques and only learns on benign policy data;
- The methodology creates an embedding representation of every policy node in the graph.

Overview. In Figure 2, we illustrate the flow of our proposed approach during the testing phase, which consists of three main stages: graph creation, graph embedding, and anomaly detection. The goal of the approach is to detect potential misconfigurations by being deployed in a cloud environment. The first stage is to build a graph model from the identity and access management policies. The second stage is graph embedding. In this stage, the graph model is transformed into a vector space, while maximally maintaining graph information and structure. Finally, anomaly detection is performed on new observations of policies in the embedded vector space to detect outliers, and therefore potential misconfigurations. If any outliers are detected, an alert is raised.

Our proposed approach operates in two phases: the training phase and the detection phase. The flow of the detection phase has been described above. The flow of the training phase is similar but has two important distinctions. First, only benign policy data is used in this phase, this means the anomaly detection model is only trained on properly configured policies. Policies are considered to be properly configured if they follow security policies, adhere to industry best practices, or are configured using existing solutions to ensure proper configuration. Second, instead of using the anomaly detector to evaluate new observations, the anomaly detector is trained and the model is created.

Next, we describe each stage of our proposed approach in more detail.

Graph Creation. For the creation of the graph, we define a cloud instance as c, where the graph representation of the policies is noted as $G(c)$. The graph consists of vertices and edges noted as (V, E). This gives us the following graph representation of the policies in the cloud as: $G(c) = (V, E)$. Each node V in the graph is of one of the following types: policy, action, resource, or entity. Where entity can be one of the following: user, group, or role. Policy nodes represent the actual identity and access management policies in the environment. Action nodes represent the specified actions within the permissions in the policy, while resource nodes represent the resources on which the actions apply. Entity nodes are the identities that are present within the environment and can make use of the policies.

The relationships between the nodes are defined as edges E in the graph as follows:

```
(Entity)-[ATTACHED_TO]->(Policy)
(Policy)-[ALLOWS | DENIES]->(Action)
(Action)-[WORKS_ON]->(Resource)
```

Graph Embedding. Graph models are not well suited to be used in machine learning models. This is due to the fact that working with graph models is computationally very expensive and transferring the information contained in graphs to machine learning models is hard. Therefore, we first need to transform the graph. In our, approach we use a technique called graph embedding [20]. With graph embedding, we are able to transform our previously created graph into a vector representation, while still maximally maintaining important information about the graph and graph structure. Information that is maintained can for example be the number of actions and resources in the policy or the attached entities.

Graph embedding enables the model to learn continuous feature representations of the nodes in the graph. It learns a mapping of nodes to a low-dimensional space of features that maximizes the likelihood of preserving graph neighborhoods of nodes. Therefore, nodes that are similar and close to each other in the graph, also remain similar and close to each other in the vector representation. Node similarity can be seen as the level of permissiveness they have, so the number of actions and resources contained in the policy, but also the entities that are attached to the policy. The graph embedding is based on random path sampling through the graph, which maximizes the neighborhood exploration.

The graph embedding technique is applied on the policy nodes in the graph. This is because the goal is to detect potential misconfigurations at the policy level.

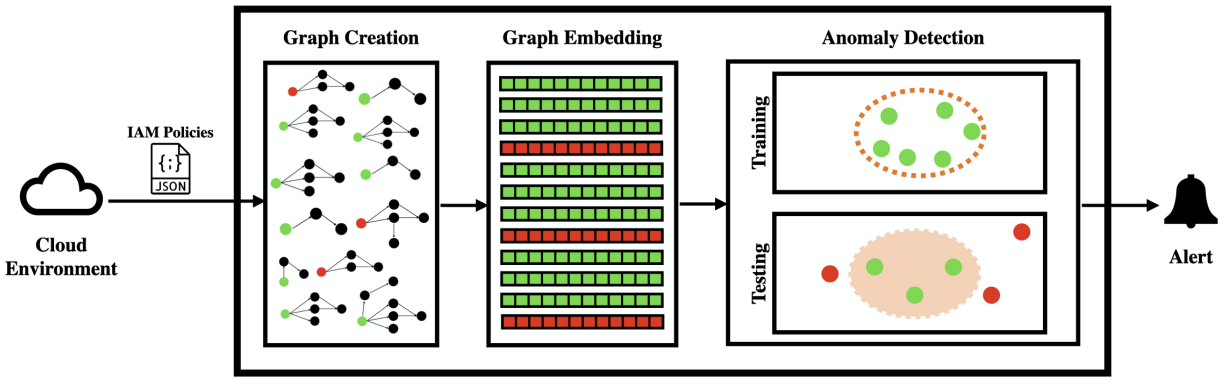


Figure 2: Overview of our proposed approach to detect potential misconfigurations. First, we create graph model representations of the IAM policies in the cloud environment (Graph Creation); then, we transform the graph model into vector representations (Graph Embedding), and finally, we apply anomaly detection to spot outliers and therefore detect potential misconfigurations.

Because the other nodes are connected to the policy by definition, they are considered in the graph embedding.

Anomaly Detection. The final stage in our proposed approach is to use anomaly detection to detect whether the embedding of a policy node is a potential misconfiguration. We assume that similar policies also have similar graph representations. Therefore, properly configured policies are similar to each other, and misconfigurations are different. Based on this assumption, our main insight is that we can use anomaly detection techniques to spot outliers and therefore detect potential misconfigurations. The anomaly detection model is trained on properly configured, and therefore benign data. These can be policies that follow security policies, adhere to industry best practices, or are created through the use of existing tools to ensure proper configuration. Then, when new observations occur, they are checked against the model, if they are outliers, they are marked as potential misconfigurations. This enables our approach to monitor cloud environments over a period of time since misconfigurations can be introduced every time a change is made.

During the experimental evaluation of our proposed approach, we have evaluated several anomaly detection algorithms, the findings will be discussed in Section 5. Based on these findings and the aforementioned features, our proposed approach uses Local Outlier Factor [21]. LOF is a density-based model, where a data point is considered an outlier if it has a lower local density compared to its neighbors. The model does not make any assumption on the probability distribution of the data, nor it attempts to divide the data using a single curve, which makes it suitable for our approach.

4. IMPLEMENTATION

While in the core our proposed system is not dependent on a specific cloud provider, we have decided to focus on the AWS cloud platform during the development. This due to the fact that AWS is currently still the most popular and largest cloud provider [22].

Graph Creation. For the graph creation stage, we make use of the graph database management system Neo4j [17]. Neo4j is an open-source platform that enables us to create and interact with graph databases. The platform relies on the Cypher Query Language, which allows for the easy creation of nodes and their connecting edges.

To initially create the graph, we first create a node for every policy. From the policies, we extract the action and resource fields and create nodes for them as well. When the nodes are created, we define the relationship between the policy, actions, and resources. Finally, we create nodes for the entities and attach them to the policies that they can make use of. An example of such a graph can be seen in Figure 3. Where the yellow node is the policy, blue nodes are actions, and the purple node is the resource.

Existing graphs can also be updated if changes are made to the policies in the environment. This is done by either adding new nodes or edges, deleting existing nodes or edges or by modifying existing node properties. This does not require the recreation of the whole graph.

Graph Embedding. The second stage of our approach is graph embedding, which is as explained needed to transform a graph to vectors which we can then use in machine learning models. To do this, we have used the algorithm Node2vec [23]. This algorithm is con-

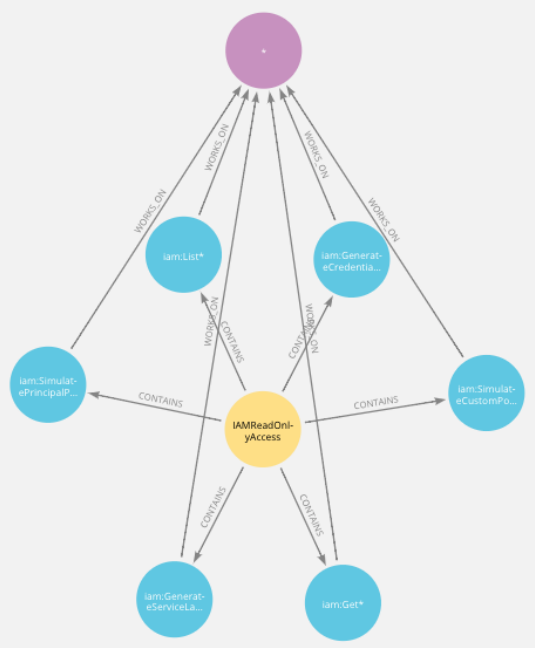


Figure 3: Example of a graph representation of a policy in the environment

sidered to be state-of-the-art in transforming graphs into feature vector representations. Another major advantage of node2vec is that it is built-in in the Neo4j platform. Therefore, using it on our created graphs is very optimized. To use the algorithm, a call, with the specification of parameters, can be made on the whole database. We have used the default parameters for node2vec, where the most important one is the dimension of the embedded vectors, which is 128. This parameter has been chosen because it is currently considered industry best practice.

After running the node2vec function on the graph, the embedded feature vectors are saved as node properties in the policy nodes. This means that we can access the embedded vectors by querying the database. This enables us to extract the vectors for the anomaly detection step.

Anomaly Detection. The final stage of our approach is anomaly detection. For the implementation of the anomaly detection techniques, we make use of the scikit-learn Python tool kit [24]. This kit provides us access to a wide range of machine learning algorithms, including the Local Outlier Factor algorithm and other needed algorithms. More details on the implementation of the anomaly detection will be discussed in Section 5

5. EVALUATION

To evaluate our proposed approach, we seek to answer the following research questions:

RQ1: What is the best performing anomaly detection algorithm to detect potential misconfigurations?

RQ2: Are the optimal parameters of the anomaly detector transferable between datasets?

RQ3: What is the overall detection performance of our proposed system in detecting potential misconfigurations?

To answer the above research questions, we designed and performed several experiments on real-world data. Next, we describe our experiments and their outcomes.

5.1 Datasets

In order to answer the above research questions, we use real-world identity and access management data of three AWS cloud environments. To collect the needed data, we have implemented a data collection Python program. The collector uses the AWS CLI [25] to interact with the cloud environment and is accessible on Github [26]. The collector collects all the identity and access management policies, and all the entities (user, groups, and roles). When the data is collected, it is exported in an Excel file. Before the export, we anonymize all the sensitive information by using a secure cryptographic hash function.

The composition of the three datasets can be found in Table 1. The first two datasets belong to two different Dutch enterprises. Dataset 1 belongs to a Dutch financial company and has approximately 12,000 employees. Dataset 2 belongs to another Dutch financial company with approximately 130 employees. Both companies are clients, which enabled us to use their cloud environments. The third dataset belongs to a smaller Italian company, with only 4 employees. The company was willing to let us collect their data due to their interest in our project.

Dataset	# of policies	# of users	# of groups	# of roles	# of collections
1	842	0	0	55	8
2	812	0	0	34	2
3	826	2	1	10	12

Table 1: Composition of three collected datasets used for the validation

As one can see in Table 1, the first two datasets do not contain any users or user groups. This is because in those environments users are being federated. This means they authenticate using another identity provider, through which they receive temporary credentials to use in the cloud environment, which they can use to assume roles. This is a common industry practice and does not impact our approach. The graph models are created with only roles and therefore the anomaly detectors only use roles as well.

Furthermore, all three datasets have at least 2 points of data collection, this can also be seen in Table 1. This means that changes to the policies have also been captured, and we can verify whether misconfigurations have been introduced at a later point in time, after the initial configuration.

Data Labeling. In order to evaluate our proposed approach, we need labeled data. Therefore, we have manually labeled all three datasets for benign policy data and potential misconfigurations. Each policy has been manually reviewed for the level of permissiveness. Policies that contain a high number of allowed actions, on a high number of resources, were considered with extra care. Each policy has been compared against the current industry best practices [27], and policies that were over permissive have been labeled as potential misconfigurations. An example of such a potential misconfiguration is the previously mentioned AdministratorAccess policy. This policy has a high level of permissiveness since it grants permission to perform all the actions on all the resources. When not attached to the proper entities this policy could be a potential misconfiguration. All three datasets are labeled, and respectively contain 12, 11, and 6 potential misconfigurations.

We also reviewed the changes that have been made to the policies throughout the different collections that we have performed. However, due to the limited access and collections, no new misconfigurations were introduced in the modifications. Therefore, we will simulate the temporal aspect in our experiments.

5.2 Experiments

Best Suited Anomaly Detection. For the first experiment, our goal was to answer **RQ1**. What is the best performing anomaly detection technique for our proposed approach? To answer this question, we focus on one dataset, the most complete one, which is dataset 1. This dataset contains the most policies and roles. To answer the question, we follow our proposed approach as described in Section 3.

First, we create graph model representations from the first IAM policy data collection, using the Neo4j graph database platform. After the graph model is created, we apply graph embedding to transform the policy nodes into vector representations. Before we apply the anomaly detection techniques, we need to prepare a training and testing set.

Dataset 1 contains a total of 842 policies and we have spotted 12 misconfigurations. To ensure that the anomaly detection model is solely trained on benign data, we temporarily remove the misconfigurations from the dataset. This leaves us with 830 benign policies. From 90% of the benign policies, we create a training set, therefore our training set size is 747 benign policies. The test set consists of the remaining 10% of benign policy data, or 83 policies, and the 12 misconfigurations. The total test set size is therefore 95 policies. We choose a 90/10 training testing split, because of the nature of IAM policies in cloud environments. Most of the policies are created initially when the environment is created. New policies are added through the life-cycle of the environment, but only comprise a small part of the total amount. Furthermore, the test set is imbalanced, there are considerably more benign policies than misconfigurations. This also represents the real world, since misconfigurations are introduced much less frequently, than benign modifications. This simulates the temporal of cloud environment changes. The training set can be seen as the cloud environment at the beginning. The test set is then the changes made to the policies in the environment, both additions of policies as well as modifications, and can therefore be considered new observations.

In order to evaluate the effectiveness of our proposed approach, we implement 4 anomaly detection algorithms, namely:

- One-Class Support Vector Machine
- Local Outlier Factor
- Isolation Forest
- Robust Covariance

Each anomaly detector goes through the same process. We first train the model using the benign training set. Then, we evaluate the performance of the model using the test set, which contains both benign policies and misconfigurations. To evaluate the performance we use the following metrics: precision, recall, F1-score, and ROC Area Under Curve. Precision is used to measure the proportion of positive identifications that were actually correct. Recall is used to measure the proportion of actual positives that were identified correctly. F1-score is the harmonic mean of precision and recall, and conveys the balance between the precision and the recall. ROC Area Under Curve (ROC AUC) is used to measure the relationship between the True Positive and False Positive Rate of the anomaly detector.

To maximize the performance of the anomaly detectors we apply parameter optimization. Parameters are internal model variables that determine the performance of the model. By optimizing the parameters, we maximize the performance of the model. The effect of the parameters on the performance metrics can be found in Figure 4. The metrics which we optimize for are ROC AUC, precision, and recall. Therefore, this gives us the following optimal parameters for the anomaly detectors:

- One-Class SVM: gamma = 0.001, nu = 0.5
- Local Outlier Factor: n_neighbours = 5

- Isolation Forest: n_estimators = 30
- Robust Covariance: contamination = 0.1

After selecting the optimal parameters, we evaluate the performances of the four anomaly detection techniques. The maximized performances can be found in Table 2. The first thing that we notice, is that all four techniques have relatively high precision. For the recall, however, there are bigger differences. Both metrics are important since precision measures how well the anomaly detectors find only anomalies. While recall measures how well the detectors find all anomalies. In our approach, a high recall is more important. This is because we believe that potential misconfigurations that are missed by the model will have a considerably larger impact, than a false positive detection.

Looking at the results in Table 2, based on the recall metric there is one algorithm that stands out, namely the Local Outlier Factor (LOF). LOF has a recall of 0.69, which means that 69% of the misconfigurations are correctly spotted. This means that 8 out of 12 misconfigurations are spotted by the anomaly detector. LOF achieves a precision of 0.80, which means that 80% of all policies that are marked as potential misconfigurations, are actual misconfigurations, and 20% are false positives.

Considering the combination of precision and recall, as well as the ROC AUC, we can determine that LOF is the best performing anomaly detector for our approach. Therefore, we have answered **RQ1**.

Algorithm	Precision	Recall	F1-Score	ROC AUC
One-Class SVM	0.86	0.58	0.65	0.70
Local Outlier Factor	0.80	0.69	0.73	0.66
Isolation Forest	0.89	0.29	0.32	0.60
Robust Covariance	0.84	0.64	0.71	0.65

Table 2: Detection performance of the anomaly detectors after parameter optimization on dataset 1

Parameter Transferability. We design the second experiment to answer **RQ2**. Is the optimal parameter, that we have found in the previous experiment, transferable between datasets. To answer this question, we use datasets 2 and 3. The basis of this experiment is very similar to the first experiment. We first create separate graph models for the two new datasets, then we embed the policy nodes into vector representations. However, instead of using all four anomaly detection techniques, we only apply Local Outlier Factor as an anomaly detector. Also, instead of performing parameter optimization, we use the previously found optimal parameter, n_neighbours = 5.

We also apply the same 90/10 training testing split as before. Datasets 2 and 3 contain respectively 812 and 826 policies in total. For dataset 2 we create a training set of 737 benign policies and a testing set of 11 misconfigurations, and 64 benign policies (total test size: 75). For dataset 3 we create a training set of 743 benign policies and a testing set of 6 misconfigurations, and 77 benign policies (total test size: 83).

Now that we have the training and testing sets for both datasets, we can perform anomaly detection. For both training sets, we create a LOF model, which we train on the benign training sets, using the optimal parameter, n_neighbours = 5. Finally, we evaluate the performance of the anomaly detector using the two mixed testing sets. The results can be found in Table 3

Dataset	Precision	Recall	F1-Score	ROC AUC
2	0.84	0.77	0.80	0.73
3	0.90	0.72	0.78	0.72

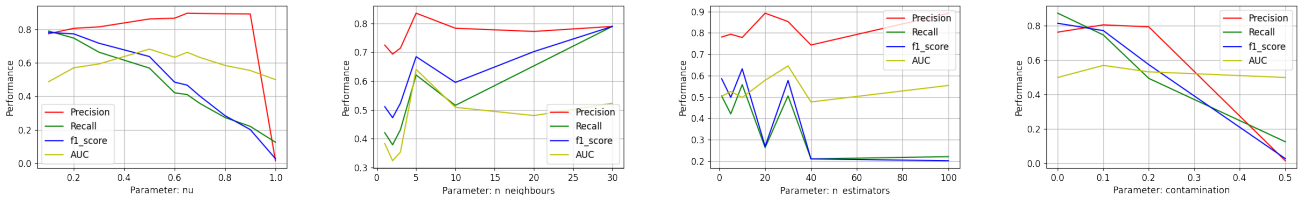
Table 3: LOF detection performance on dataset 2 and 3, with parameter n_neighbours = 5

From the results, we observe that the detection performance is even slightly higher compared to the previous results on dataset 1. All metrics for datasets 2 and 3 have improved compared to dataset 1. This shows that the found optimal parameter, n_neighbours = 5, is indeed transferable between datasets, and answers **RQ2**.

The fact that the parameter is transferable is no surprise. We believe there are two reasons for this. First, all three datasets are very similar to each other. This is because the identity and access management policies are all structured in the same way. Policies contain actions and resources and are attached to entities. This is the same for all three datasets. Second, by default, there are 515 IAM policies managed and provided by AWS. Therefore, there is a significant overlap of data between the datasets. Because of this, the model is able to spot misconfigurations because it learns on properly configured policies. This is the same, regardless of the dataset.

We do notice that the performance of the LOF anomaly detection is higher on the two new datasets. After careful consideration and analysis, we do have an intuition why this is the case. During the analysis, we notice that there is a clearer distinction possible between benign policy data and misconfigurations. We notice that dataset 1 has more policies that offer a high level of permissiveness, without being misconfiguration. In datasets 2 and 3 however, we see that in general the policies are more conservative, and therefore have a lower level of permissiveness. This enables the LOF anomaly detector to create a clearer distinction between benign data and misconfigurations and therefore enhances the performance.

Overall Detection Performance. Finally, we can answer **RQ3**, what is the overall detection performance



(a) One-Class SVM **(b)** LOF **(c)** Isolation Forest **(d)** Robust Covariance

Figure 4: Detection performance metrics during parameter optimization for the four anomaly detection techniques

of our proposed approach. In Table 4 we combine the results of the LOF anomaly detection on all three datasets.

Dataset	Precision	Recall	F1-Score	ROC AUC
1	0.80	0.69	0.73	0.66
2	0.84	0.77	0.80	0.73
3	0.90	0.72	0.78	0.72

Table 4: LOF detection performance on all three datasets

For the overall detection performance of our approach, we compute the weighted average precision and recall. As weight factor, we use the proportion of policies in the dataset to all collected policies. In total, the three collected datasets contain 2480 policies, which gives us the following weight factors: 0.34, 0.33, and 0.33. This gives us a weighted average precision of 0.85 and a weighted average recall of 0.73. In our approach a higher recall is important. Since recall measures how many actual misconfigurations are detected, and missing misconfigurations is very costly.

An average recall of 0.73 means that 73% of the misconfigurations, that otherwise would go undetected, will now be detected before leading to major problems. An average precision of 0.85, means that of all policies marked as misconfigurations, 85% are indeed misconfigurations, and 15% are false positives. The percentage is not ideal and can be improved, but it still manageable for a manual review of all potential misconfigurations. Therefore, we believe that our approach is effective in detecting potential misconfigurations.

5.3 Runtime Performance

During the experiments, we also measured the runtime performance of the graph creation and model training stage of our approach.

Graph Creation. First, we consider the graph creation stage. In this stage, the identity and access management policy data is transformed into a graph model representation. The runtime performance of this stage can be found in Figure 5. The performance scales linearly, but in the case of larger datasets can take a while. In the case of our complete dataset, the graph creation took 20 minutes and 55 seconds. It is worth mentioning that only the initial graph creation is this long, this is due to the fact that all the nodes need to be created. Graph updates are considerably faster since the number of new nodes is generally low.

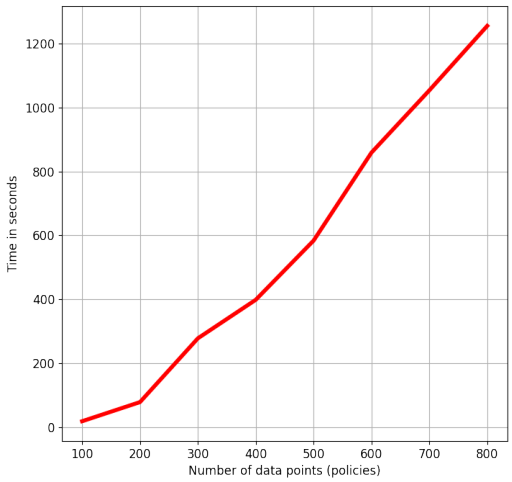


Figure 5: Runtime performance of creating a graph model from the policy data, as well as embedding the graphs into vectors.

Model Training. We also consider the model training overhead of the Local Outlier Factor anomaly detection model. The LOF model is trained on benign data and then used to determine whether new observations are also benign or potential misconfigurations. We have measured the runtime performance of the LOF model, the results can be found in Figure 6. As can be seen, LOF scales linearly for the number of policies to train on. The training times are low because of the optimized use of the vector representation of the policy data.

Furthermore, the testing of new observations is also fast. Especially because only a relatively small number of new observations happen at a time.

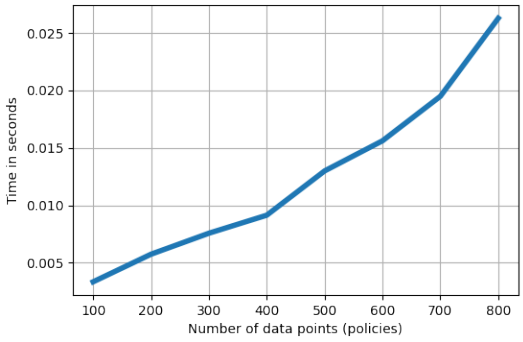


Figure 6: Runtime performance of training the LOF model

5.4 Discussion of Results

As closing remarks for this section, we discuss the results and the cause of potential misclassifications in the approach. First of all, we revisit the goals for our approach. Our goals were to develop a fully automated, proactive, generic, and requiring a low effort and maintenance approach. We believe that our proposed approach fulfills these goals. The proposed approach is fully automated and does not require effort or maintenance. After deployment, no user interaction is needed to make the approach work. The approach is proactive since it detects potential misconfigurations before they can be abused. Finally, the approach is also generic, as it does not require specific knowledge of misconfigurations. The approach learns a model based on properly configured policies and checks whether new observations adhere to the model. Therefore, all goals for our approach have been fulfilled.

In our approach, false positives and false negatives can occur. After investigating the cause of this, we have found that false positives can occur on policies that have a high level of permissiveness. This means that the policies permit a high number of actions on a high number of resources. Even if the policy is properly configured, and the allowed actions are restricted, it could still be misclassified as a potential misconfiguration. An example of such policy is the *ReadOnly* policy. This policy permits 762 read-only actions, which makes it very permissive. However, since they are read-only permissions on non-critical resources, we do not consider this a misconfiguration, but the anomaly detector still classifies it as a potential misconfiguration.

A similar situation happens in the case of false negatives. False negatives can occur when a policy does not seem permissive but actually allows certain high-impact actions. An example of such a policy is the *PowerUserAccess* policy. This policy only allows a small number of actions, but they work critical resources and can have a high impact.

Further research is needed to further improve the detection performance, which we will discuss in the following sections.

6. LIMITATIONS

In this section, we discuss some limitations of our proposed approach.

Benign Policy Data. First and foremost, a limitation of our approach could be the assumption that we only train the anomaly detection model on benign data. This is because the policy data that has been collected is from active and used cloud environments. Each dataset has been manually reviewed, and potential misconfigurations have been removed from the training set. There are two limitations to this approach. First, the definition of what a misconfiguration is. In our approach we have followed industry best practices to spot potential misconfigurations, this could however mean that during manual labeling, misclassification mistakes have been made. Second, it is still possible that some misconfigurations have gone unnoticed and ended up in the training set. We believe that due to very careful review, the number of misconfigurations that have slipped through is sufficiently low, to not have had a major impact on the training.

Advanced Embedding and Anomaly Detection. In our approach, we have used the graph embedding technique Node2vec, which is currently in the state-of-the-art of graph embedding. Node2vec in combination with LOF has already provided us with good results in detecting potential misconfigurations. There are however newer techniques in the making that might be able to transform the graph in a better and more efficient way. An example of such a new embedding technique is GraphSage [28], which uses inductive representation learning to also enable the embedding of node properties.

Furthermore, we have only considered four anomaly detection algorithms in our approach. More complex machine learning techniques could further enhance the performance of our approach. Examples of such techniques are Graph Convolutional Networks [29], and One-Class Neural Networks [30].

Resource-Based Policies. In our current approach, we only consider identity-based policies. There are, however more identity and access management policies in cloud environments. For example, there are also resource-based policies. These are policies that are not attached to entities in the environment, but rather directly to the resource. Therefore, specifying

actions that are only allowed on that specific resource, regardless of who is performing the action. An example of a resource-based policy is setting a storage service to be either publicly or privately accessible. Resource-based policies are created and stored in the same way as identity-based policies.

7. FUTURE WORKS

Apart from the limitations discussed in the previous section, which will be addressed, we also see several future works to further improve our proposed approach.

First, by adding a feedback loop in the future, the system will be able to handle false positives better. When a false positive occurs, or in case the raised alert is intended behavior, the model will then be able to learn from this and prevent another similar alert in the future.

As we mentioned before, our current focus has been on the AWS cloud platform. Although, we believe that our proposed approach is not limited to just one cloud platform. Therefore, future work is to evaluate the performance of our approach on cloud environments of other cloud providers.

Finally, link prediction techniques could be added to the system. This will allow the system not only to detect potential misconfigurations but also to detect missing policies for entities that should have the permissions. This will allow for remediation before it leads to problems. Link prediction is possible with the techniques used in our approach but is currently out of scope.

8. RELATED WORKS

Access Control is a subfield of the broader area of identity and access management and has been studied extensively. A number of tools have been proposed to detect misconfigurations in access control systems. These tools can be divided into two categories, internal detection, and external detection tools.

First, we consider the internal detection solutions. P-Diff [16] is a tool for monitoring access and control behavior by using decision tree algorithms. While being effective, there are two major limitations, First, the tool learns access control policies from access logs, and therefore is limited to the information contained in the access logs. Second, the approach can be considered reactive, since the misconfiguration will only be detected if it shows up in the access logs.

The second tool is Baaz [31]. Baaz infers permission misconfigurations in an enterprise network my monitoring updates made to the access control metadata, and looking for potential inconsistencies among peers. Meaning that similar users, should also have similar permissions, if this is not the case, it might be a potential misconfiguration. The major limitation of Baaz is that it relies on the definition of what should be considered as an inconsistency. This parameter can be tweaked by administrators, but could still cause problems and influence the performance of the system.

The second category consists of external detection tools. A research study has been performed by Continella et al. [32], where the authors focused on Amazon Simple Storage Service (S3) buckets [8]. The authors focused on identifying misconfigurations, more specifically key misuse, and permission configurations, that affect users' privacy and security. The most important limitation of this work is that it only focused on the S3 storage service. Another research has been performed on the cause of data leaks when cloud platforms are used as mobile app back-ends [33], resulting in a tool called LeakScope. The tool consists of three key components: Cloud API identification, String Value Analysis, and Vulnerability Identification. The tool has been proven to be successful, however, some limitations have still been found. First, it has false negatives, this is because the list of used APIs might be incomplete, and therefore certain aspects can be missed. Secondly, the String value analysis tool is not able to recognize dynamically generated values. Furthermore, the ethical implications posed as a limit in this research.

Rule-Based Solutions are the second category of solutions. This approach relies on predefined rules to which the newly created or modified cloud resources need to adhere to. These rules have to be created, monitored, and maintained throughout the life cycle of the cloud environment. The set of defined rules are usually created to be in line with the company security policies. There are several existing (open source) solutions that implement this rule-based approach for detecting security misconfigurations.

Cloud Custodian [15] is a widely used open-source rule-based system. Cloud Custodian enables users to be well managed in the cloud. It allows for an easy definition of rules to manage the cloud infrastructure. These rules are collected in policies. The policies can be as simple or as complex as the person creating them wants them to be. Examples of such policies can be the blocking of all the public access to S3 buckets or the detection of an account receiving admin privileges. AWS Remediation Framework [34] is another example of an open-source solution. As the name suggests, it is a project that identifies and remediates AWS security issues to ensure AWS usage is compliant with a set of rules.

Although these rule-based solutions can be very powerful and have clear advantages, there are also a number of limitations. First of all, the rules need to be created and maintained to adhere to security policies. This has to be performed manually and can require a large ef-

fort. Furthermore, this process can be error-prone, and security policies can be insufficient to detect all misconfigurations. Secondly, cloud environments are generally extremely dynamic and change frequently. There are situations in which a certain action can be seen as a misconfiguration, while it is needed for a certain operation, predefined rules can therefore be too rigid to handle these quick changes, which will impact the performance of the system.

Cloud-Native Solutions are the final category. Cloud providers have also started offering solutions for detecting misconfigurations. AWS provides CloudTrail [35], which is an AWS service that enables governance, compliance, and auditing of the AWS account and all the corresponding resources. It provides logging and continuous monitoring of the AWS environment. Cloudtrail can be used in two ways to detect misconfigurations. First, it can be used to log and raise alerts in case any changes are made to the identity and access management configurations of the cloud resources. Second, it can be used to detect unauthorized access if misconfigurations are abused. Both ways have limitations. Either over-alerting administrators on every change made, or reacting to already happened abuse, thus being too late.

Furthermore, AWS has some mechanisms in place to prevent misconfigurations. For example, when overly permissive identity and access management roles are created, the system raises a warning. This already creates the first line of defense, however, it can be easily overridden by the administrator, and only identifies major and obvious errors.

Anomaly Detection on graph-based models is becoming widely used. Therefore, we also collected several related works in this field. This technique has been extensively discussed in the survey on graph-based anomaly detection [36]. In this survey, the authors provide a general, comprehensive, and structured overview of the state-of-the-art methods for anomaly detection in data represented as graphs. They prove that graphs are very capable of capturing data, and current anomaly detection techniques can be very powerful.

Secondly, we consider ProvDetector [37], which is a provenance-based approach for detecting stealthy malware, where anomaly detection techniques are applied on provenance graphs to detect malicious behavior. The approach is similar to our approach. First, they create a graph of the system calls, this graph is embedded and finally, anomaly detection is applied. Although the approach is similar to ours, it is in a completely different field.

Finally, we consider graph-based anomaly detection approaches for fraud detection [38]. These approaches are among the most popular techniques used to analyze connectivity patterns in communication networks and identify suspicious behaviors. Again, this is in a different field, but there are two important things to consider from this work: first, one should avoid feature engineering but, instead consider a range of graph representation learning techniques; and second, consider methods to automate feedback for continuous model learning.

9. CONCLUSION

In this paper, we presented a novel approach for detecting misconfigurations of AWS identity and access management policies. The goals for this approach were to be fully automated, proactive, generic, and requiring low effort and maintenance. To achieve this, our proposed approach first creates a graph model from the identity and access management policy data. The graph model is then embedded into vector representations of the policy nodes. Finally, we train an anomaly detection model on benign policy data. Each new observation is then compared against the model and classified as either benign data or potential misconfiguration. We have evaluated our approach on real-world data from three active cloud environments. The results show that our approach has a relatively high precision and recall for the evaluated policy data, demonstrating the effectiveness of our proposed approach. Furthermore, we have shown that optimal parameters for the anomaly detection algorithms are transferable between environments, while still maintaining a similar detection performance.

Although we have demonstrated the effectiveness of our proposed approach, there are still improvements to be made in the future. Newer embedding and anomaly detection techniques could be investigated. Furthermore, our focus has been on AWS, but the proposed approach can be applied to other cloud environments.

Security misconfigurations are a big problem, especially in cloud environments. We believe our approach is a step in the right direction to mitigate the risk of security misconfigurations in the future.

10. REFERENCES

[1] L. Whitney, "2020 sees huge increase in records exposed in data breaches," Jan 2021. [Online]. Available: https://www.techrepublic.com/article/2020-sees-huge-increase-in-records-exposed-in-data-breaches/

[2] E. Flitter and K. Weise, "Capital one data breach compromises data of over 100 million," Jul 2019. [Online]. Available: https://www.nytimes.com/2019/07/29/business/capital-one-data-breach-hacked.html

[3] T. Seals, "Millions of hotel guests worldwide caught up in mass data leak." [Online]. Available:

https://threatpost.com/millions-hotel-guests-worldwide-data-leak/161044/
[4] ——, "Good heavens! 10m impacted in pray.com data exposure," Nov 2020. [Online]. Available: https://threatpost.com/10m-impacted-pray-com-data-exposure/161459/
[5] "Misconfigured aws s3 bucket leaks 36,000 inmate records," Feb 2020. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/misconfigured-aws-s3-bucket-leaks-36-000-inmate-records
[6] Nag, "S3-leaks." [Online]. Available: https://github.com/nagwww/s3-leaks
[7] Pbnj, "Yet another s3 bucket leak." [Online]. Available: https://github.com/pbnj/YAS3BL
[8] AWS, "Amazon simple storage service (amazon s3)," 2002. [Online]. Available: https://aws.amazon.com/s3/
[9] J. Brook, A. Getsin, G. Jensen, L. Jameson, M. Roza, N. Thethi, A. Kurmi, S. Levy, S. Shamban, V. Hargrave *et al.*, "Top threats to cloud computing: The egregious eleven," *Cloud Security Alliance*, 2019.
[10] C. Dietrich, K. Krombholz, K. Borgolte, and T. Fiebig, "Investigating system operators' perspective on security misconfigurations," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1272–1289.
[11] M. Kazim and S. Y. Zhu, "A survey on top security threats in cloud computing," 2015.
[12] I. M. Khalil, A. Khreishah, S. Bouktif, and A. Ahmad, "Security concerns in cloud computing," in *2013 10th International Conference on Information Technology: New Generations*. IEEE, 2013, pp. 411–416.
[13] H. G. Buff, "Compliance," 2000. [Online]. Available: https://aws.amazon.com/compliance/shared-responsibility-model/
[14] K. W. Bennett and J. Robertson, "Security in the cloud: understanding your responsibility," in *Cyber Sensing 2019*, vol. 11011. International Society for Optics and Photonics, 2019, p. 1101106.
[15] "Cloud custodian." [Online]. Available: https://cloudcustodian.io/
[16] C. Xiang, Y. Wu, B. Shen, M. Shen, H. Huang, T. Xu, Y. Zhou, C. Moore, X. Jin, and T. Sheng, "Towards continuous access control validation and forensics," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 113–129.
[17] "Graph database platform: Graph database management system: Neo4j," May 2021. [Online]. Available: https://neo4j.com/
[18] AmazonWebServices, "Aws iam," 2003. [Online]. Available: https://aws.amazon.com/iam/
[19] S. Omar, A. Ngadi, and H. H. Jebur, "Machine learning techniques for anomaly detection: an overview," *International Journal of Computer Applications*, vol. 79, no. 2, 2013.
[20] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.
[21] "Novelty detection with local outlier factor (lof)." [Online]. Available: https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_novelty_detection.html#:~:text=TheLocalOutlierFactor(LOF,withrespecttoitsneighbors.
[22] K. Stalcup, "Aws vs azure vs google cloud market share 2020: What the latest data shows," Aug 2020. [Online]. Available: https://www.parkmycloud.com/blog/aws-vs-azure-vs-google-cloud-market-share/
[23] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.
[24] "Scikit-learn python tool kit." [Online]. Available: https://scikit-learn.org/stable/
[25] "Aws command line interface," 2006. [Online]. Available: https://aws.amazon.com/cli/
[26] N. Khasuntsev, "Github - cloud misconfigurations (source code)." [Online]. Available: https://github.com/utwente-scs/iam-collector
[27] AWS, "Aws iam best practices," 2003. [Online]. Available: https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html
[28] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *arXiv preprint arXiv:1706.02216*, 2017.
[29] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[30] R. Chalapathy, A. K. Menon, and S. Chawla, "Anomaly detection using one-class neural networks," *arXiv preprint arXiv:1802.06360*, 2018.
[31] T. Das, R. Bhagwan, and P. Naldurg, "Baaz: A system for detecting access control misconfigurations." in *USENIX Security Symposium*, 2010, pp. 161–176.
[32] A. Continella, M. Polino, M. Pogliani, and S. Zanero, "There's a hole in that bucket! a large-scale analysis of misconfigured s3 buckets," in *Proceedings of the ACM Annual Computer Security Applications Conference (ACSAC)*,

December 2018.
[33] C. Zuo, Z. Lin, and Y. Zhang, "Why does your data leak? uncovering the data leakage in cloud from mobile apps," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1296–1310.
[34] Flatironhealth, "Aws remediation framework." [Online]. Available: https://github.com/flatironhealth/aws-remediation-framework
[35] "Aws cloudtrail." [Online]. Available: https://aws.amazon.com/cloudtrail/
[36] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data mining and knowledge discovery*, vol. 29, no. 3, pp. 626–688, 2015.
[37] 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.
[38] T. Pourhabibi, K.-L. Ong, B. H. Kam, and Y. L. Boo, "Fraud detection: A systematic literature review of graph-based anomaly detection approaches," *Decision Support Systems*, vol. 133, p. 113303, 2020.