# A Model-based Approach to Anomaly Detection in Software Architectures

Hemank Lamba, Thomas J. Glazier,
Bradley Schmerl, Javier Cámara, David Garlan, Jürgen Pfeffer
Institute for Software Research
Carnegie Mellon University
Pittsburgh, PA, USA
{hlamba,tglazier,schmerl,jcmoreno,garlan,jpfeffer}@cs.cmu.edu

## ABSTRACT

In an organization, the interactions users have with software leaves patterns or traces of the parts of the systems accessed. These interactions can be associated with the underlying software architecture. The first step in detecting problems like insider threat is to detect those traces that are anomalous. In this paper, we present a method to find anomalous users leveraging these interaction traces, categorized by user roles. We propose a model based approach to cluster user sequences and find outliers. Such a technique could be useful in finding potentially anomalous users, insiders, or compromised accounts.We show that the approach works on a simulation of a large scale system based on and Amazon Web application style.

## Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

## Keywords

anomaly detection, model-based graph clustering

## 1. INTRODUCTION

In the contemporary corporate setting, having anomalous users interacting with software is potentially disastrous and can cost organizations millions of dollars. For instance, Cummings et al. [12] studied 80 cases of fraud in the financial services sector. They found that in most cases, the users interacted with systems they had access to, due to their roles in the companies – however, they were behaving anomalously, compared to how they had behaved before, or how others in similar roles behaved. They discovered that the impact of such cases was as high as 28 million dollars. Similar results were found in other sectors as well - government [29], the information technology and telecom sector [30] and critical infrastructure sectors [26]. Anomalous behavior can be caused by the existence of compromised user accounts, rogue users, or by less knowledgeable users who break things at random.

Hawkins defines an *anomaly* as "an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism" [20]. Similarly, Johnson defines an anomaly as "an observation in a dataset which appears to be inconsistent with the remainder of that set of data" [25]. Many challenges exist in identifying anomalous behavior since the number of anomalous activities is much fewer compared to the number of usual activities (making it like finding a needle in a haystack). The scale of modern systems, including the number of components they comprise and the diverse and large set of users that they are required to interact with, make manual inspection simply infeasible.

Automated approaches are therefore needed to analyze user actions and determine which of them are anomalous. To be applicable, anomaly detection algorithms need to work for modern and common systems, and ideally need to (a) scale to the size of systems and number of users, (b) accurately detect anomalies in the presence of diverse and nuanced user roles, (c) account for the complex structure and organization of today's systems, (d) understand that users have complex interactions with software systems that involve their interacting with multiple parts of the system, and (e) be interpretable at a level that enables system administrators to determine which anomalies are likely to represent malicious or dangerous behavior.

There has been considerable work in anomaly detection to try and meet these requirements with varying degrees of success. There are two general approaches: graph-based anomaly detection algorithms and activity-based algorithms. Graph-based approaches analyze organizational structures (e.g. ego-networks of nodes, communities, and subgraphs). Altneratively, activity-based methods focus on user behavior (e.g. number of logins, file access).

None of these approaches consider the possibility of characterizing the interaction of users with the software system as sequences of interaction events with its constituent components. Moreover, the approaches do not consider that organizational roles are often difficult to define and do not have precise boundaries. This scenario demands an approach that applies to complex software eco-systems and overcomes the aforementioned limitations.

In this paper, we describe an approach that improves upon

*HotSoS 2016 Pittsburgh, PA, USA*

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

existing approaches, since: (a) it can automatically infer organizational roles based on traces of user behavior through complex software systems, (b) it can represent complex systems at a software architecture level of abstraction, enabling scalability and interpretability, and (c) it can be used in the context of modern, distributed software systems of large scale with many users.

The key ideas behind our approach are:

- Use the architectural description of software systems as the primary graph on which to detect anomalies. This captures the idea that software systems are not monolithic entities, and that some anomalies can only be detected by understanding the ways in which users interact with many parts of the system;
- A novel model-based graph clustering algorithm that takes into consideration the sequence of interactions with components in the system to determine clusters based on learned roles. This approach detects anomalies by the distance that a particular trace is from any of the clusters.

We validate our approach by illustrating how it can be used to detect anomalies on a realistic simulation of an Amazon web application architecture. We inject a hand-crafted anomalous user into the system and show that our clustering algorithm is successfully able to find the anomalous user.

The rest of this paper is organized as follows. Section2 reviews the related work. Section 3 provides an overview of our approach, and how we apply it to the problem of anomaly detection. We continue by describing an example and threat scenario in Section 4. In Section 5 we describe our model-based graph clustering algorithm, and the experimental setup and results in Section 6. Finally, in Section 7 we close by discussing limitations and future work.

## 2. RELATED WORK

In the past few years, a lot of work has been done in the field of graph-based anomaly detection. Koutra et al. present a comprehensive survey on graph-based anomaly detection techniques [4], which can be classified according to the type of input graph. Specifically, the classification is generally made according to the availability of: (i) multiple snapshots of the graph, and (ii) edge/node labels. We present a brief summary of results on existing techniques [1] in Table 1.

For single snapshot graphs with no labels, the approaches are either structure-based or community-based. Structure-based approaches generally rely on generating structural features (e.g., subgraphs, cliques, ego-networks) and then finding anomalous nodes/edges based on distances between the points in the generated feature space [3, 22]. Community-based methods find densely connected groups (i.e., *communities*) and they classify anomalies as nodes/edges connecting two communities [44, 11, 47, 48, 42]. In single snapshot graphs with attributes over the nodes/edges, the approaches are similar. However, for structure-based approaches, the feature space is enhanced due to the presence of attributes/labels over nodes and edges [33, 31]. Similarly, community-based approaches take into consideration the attributes over nodes and edges and define anomalies as nodes whose attribute values differ significantly from that of its community [16, 32, 35]. For cases in which multiple snapshots of the graph at different time instants are available, the approaches proposed in the literature are feature-based, community-based and decomposition-based. Feature-based approaches come up with a metric for each snapshot of the graph, and then, based on the distance between two consecutive snapshots, flag a snapshot as anomalous. Different approaches in the literature try various metrics to better represent the snapshot [41, 7, 18, 36, 6, 28, 34]. Decomposition-based methods involve matrix or tensor decomposition. Each of these approaches compute the reconstruction error, which is the distance between the original matrix and the estimate from the decomposition. The reconstruction errors are monitored to conclude if a particular snapshot is anomalous or not. Matrix-based methods use SVD [2, 23], NMF [37], CUR [13, 46] decompositions, while tensor-based methods use STA (streaming tensor analysis) [45] and PARAFAC [27, 5] to decompose tensors. Community-based approaches keep track of the communities in the graph and indicate if there are structural or contextual changes. Examples of such approaches include GraphScope [43], GOutlier [1], Bayesian approach [21], ECOutlier [19].

Another landmark work in the area of detecting anomalous users is by Ted et al. [39]. The authors create a system that combines structural and semantic information and an ensemble of algorithms to detect cases of insider threats on a corporate database of monitored activities. The features used were based on the emails, file accesses, group information, login information, printer logs, URLs visited, and other combinations of features. The algorithms were created for six different threat scenarios. The algorithms were relational pseudo anomaly detection, relational density estimation, gaussian mixture models, ensemble gaussian mixture models, repeated impossible discrimination ensemble, cross prediction, grid-based fast anomaly detection given duplicates, vector space models, temporal based anomaly detection, community detection, streaming community detection, seed set expansion, and betweenness centrality monitoring for streaming graphs. The last few methods are graph-based methods that work on the efficient graph data structure STINGER [14]. However, in this work most of the features are *flat* (i.e., they were created based on individual activities without taking into consideration the sequence of activities followed by users).

All existing works focus on using a feature-based or a community-based approach to identify anomalous users. Moreover, all of the above mentioned methods assume that data is available in terms of either a single snapshot or multiple snapshots of the graph, and that the graph captures information about which users are connected. Our input data is different from the one employed by other techniques, since the nodes in the graph are components, based on the software architecture. We treat each sequence of user events as a path on these graphs. We propose a model-based sequence clustering approach to assign scores to every user, and based on these scores, discover anomalous user behavior. Our approach captures the full trace of user activities and tries to find anomalies in each sequence of activities. This represents an advantage to capture anomalous behavior (e.g., in the event of an attack), since a particular sequence of activities that could be considered anomalous could go

[1] We refer readers to Koutra et al. for a detailed review [4]

| Input Graph Type | Class of Method | Reference | Description |
|---|---|---|---|
| Single Graph. No labels on nodes/edges | Feature Based | ODDBALL [3] | Uses ego net based features. Spots anomalous ego-networks. |
| | | Henderson et. al. [22] | Extended the features by recursively combining local node based features. |
| | Community Based | Sun et. al [44] | Used personalized page rank scores to find communities and also spot outliers with respect to the community. |
| | | AUTOPART [11] | Transforms the adjacency matrix based on minimum description length to find communities and classifies cross-cluster edges as anomalous. |
| | | Tong et. al. [47] | Uses non-negative residual matrix factorization to spot strange connections |
| | | SCAN [48] | Uses neighborhood of vertices to club them together into communities. Those nodes that are not assigned any community are marked as outliers. |
| Single Graph with labels on nodes / edges | Feature Based | Noble et. al. [33] | It builds on frequent subgraphs and uses categorical attributes to find the most normative subgraph. The subgraphs that are away from the best substructure are marked as anomalous. |
| | | Liu et. al. [31] | Primarily designed for detecting non-crashing software bugs. Every execution is defined as a behavior graph. Frequent subgraphs are used as features for training in classification model. |
| | Community Based | CODA [16] | Unsupervised learning algorithm to find communities and spot outliers simultaneously. |
| | | GOutRank [32] | They select subgraphs and subspaces where anomalies are easily revealed and score the nodes in those spaces. |
| | | FocusCo [35] | Approach finds attribute subspace on which nodes agree, and then finds communities in this subspace. Nodes deviating from the cluster they are in are termed as outliers. |
| Multiple snapshots of a graph at different timestamps | Feature Based | Shoubridge et. al. [41], Bunke et. al [7], Gaston et. al [18] | Propose several metrics for capturing a snapshot in the sequence. Distance between two consecutive snapshots is used to classify if a particular snapshot is anomalous or not. |
| | | Pincombe et. al. [36] | The approach proposed to extract a single feature from every snapshot and use ARMA method to find anomalies in the so generated time sequence. |
| | | DeltaCon [28] | Graph-feature based similarity approach is extended for discontinuity detection. |
| | Factorization Based | Ide et. al. [23], Akoglu et. al [2] | The methods use SVD to factorize the adjacency matrix and then uses eigenvectors to reconstruct the matrix. |
| | | Rossi et. al. [37] | The proposed approach uses NMF and MDL -based role extraction algorithm to determine roles of nodes and how they change over time. Roles are taken as features and then tracked over time. |
| | | COM2 [5], Koutra et. al. [27] | Use tensor decomposition method PARAFAC to find the anomalous nodes. |
| | Community Based | GOutlier [1], ECOutlier [19], GraphScope [43] | Community detection methods are based on finding and keeping track of the communities in the network, and anomaly is found if there is any structural or contextual change. |

Table 1: Literature review of graph based anomaly detection techniques.

undetected if its constituent activities are considered individually. For example, a case of fraud could be that a user logs-in, downloads the financial database, and sends it via email to someone outside of the organization. Using existing techniques, such an activity might not be flagged as anomalous, since those approaches just capture the number of log-ins, downloads, and emails, which taken into consideration individually, are not anomalous. However, when this set of activities are taken into consideration as a sequence, they are clearly anomalous. Moreover, our approach infers the

role of the user and the expected sequence of activities pertaining to the roles. The approach outputs a score for every user, helping to discover the anomalous users and explain the extent of their anomalies.

## 3. OVERVIEW OF APPROACH

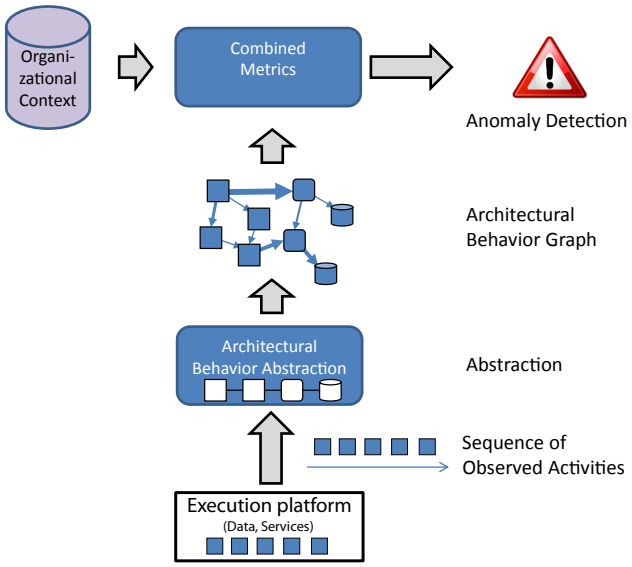



Figure 1: Overview of our approach.

In this work, we first describe the software system in the form of a graph. We use a component and connector (C&C) software architecture [40] description to obtain this graph. A component and C&C view describes the run-time configuration of the software in terms of nodes that perform computation or store data (i.e., components), and edges that represent communication between components (i.e., connectors). Secondly, we monitor the execution of these software systems to monitor user activities while they are interacting with the system, to determine interaction traces or sequences. A sequence is the order in which users interact with components along communication paths. These sequences can easily be employed to characterize paths over the underlying software architecture graph. For example, if a user accesses component 1 (e.g., a web page) which then interacts with component 2 (e.g., a server) which in turn accesses data component 3, and next component 1 again, followed by component 3, we can represent this user as the path $1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 3$. We visualize our way of converting user traces to paths on the graph in Figure 2. Once we have these paths, we apply our model-based clustering to infer the roles of these users and the transition patterns related to each of those roles. The clustering algorithm also computes an *outlier score*, which can be used by an expert later on to identify users as anomalous. We provide an overview of our approach in Figure 1.

## 4. MOTIVATING EXAMPLE AND THREAT SCENARIO

Modern software applications are built upon flexible infrastructures like public and private clouds featuring virtual machines (VMs). The elastic nature of these systems challenges traditional graph-based security methods because the individual instances of the VMs, treated as nodes in a graph, frequently change but will serve the same architectural purpose. For example, a large-scale web system will elastically add and remove VMs that are serving as web servers. Because of this architectural interchangability, a given user's path of interaction with the instances is generally predictable, despite the frequent changes in the individual VM instances. For example, a typical user request path might be web server to application server to database as determined by the logic of the code. However, a subsequent request would follow the same general path, but would likely interact with different web servers, application servers, and database servers. Therefore, a trace-based method of anomaly detection is more appropriate for cloud-based software systems.

These cloud-based software systems are commonplace in large corporations, including Target and Home Depot. Both of these organizations suffered high profile data breaches in which a compromised system was used to exfiltrate sensitive data from corporate databases and stage it on internal company servers. The data was then periodically moved by hiding it in normal network traffic. A trace-based architectural approach could have learned the normal set of interactions between the individual compute nodes and potentially identified the anomalous behavior in time to prevent the breach.

## 5. MODEL-BASED ANOMALY DETECTION

For the task of finding anomalous patterns from sequences (which are represented as paths over the software architectural graph), we propose a clustering-based framework. The clustering-based approach first clusters all the sequences into an appropriate number of clusters, and then those sequences that are weakly linked to their respective cluster are marked as anomalous or outliers.

Clustering behavior sequences is not a trivial task. Generally, two types of approaches are proposed to cluster objects – distance-based and model-based. Distance-based approaches for sequences involve computing distances between the two sequences and then use any existing clustering approach. Various ways of computing the distance between two sequences have been proposed, the most popular being counting the number of frequent patterns (n-grams) occurring in both sequences [15].

In contrast, model-based approaches try to come up with a model to explain the observed data. Most of these models have been proposed in the field of bio-informatics, where they are used to cluster DNA-sequences. It has been argued

| Time | User | Resource |
|---|---|---|
| 00:01:01 | A | 1 |
| 00:01:05 | A | 2 |
| 00:01:10 | B | 1 |
| 00:01:11 | A | 3 |
| 00:01:15 | B | 3 |
| 00:01:17 | A | 4 |
| 00:01:18 | B | 4 |

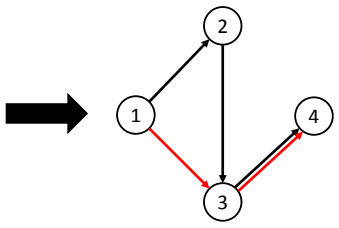



Figure 2: User traces of two users from log files modeled as paths on graphs. Nodes represent resources or components(e.g. server) and edges denote the traces of single users.

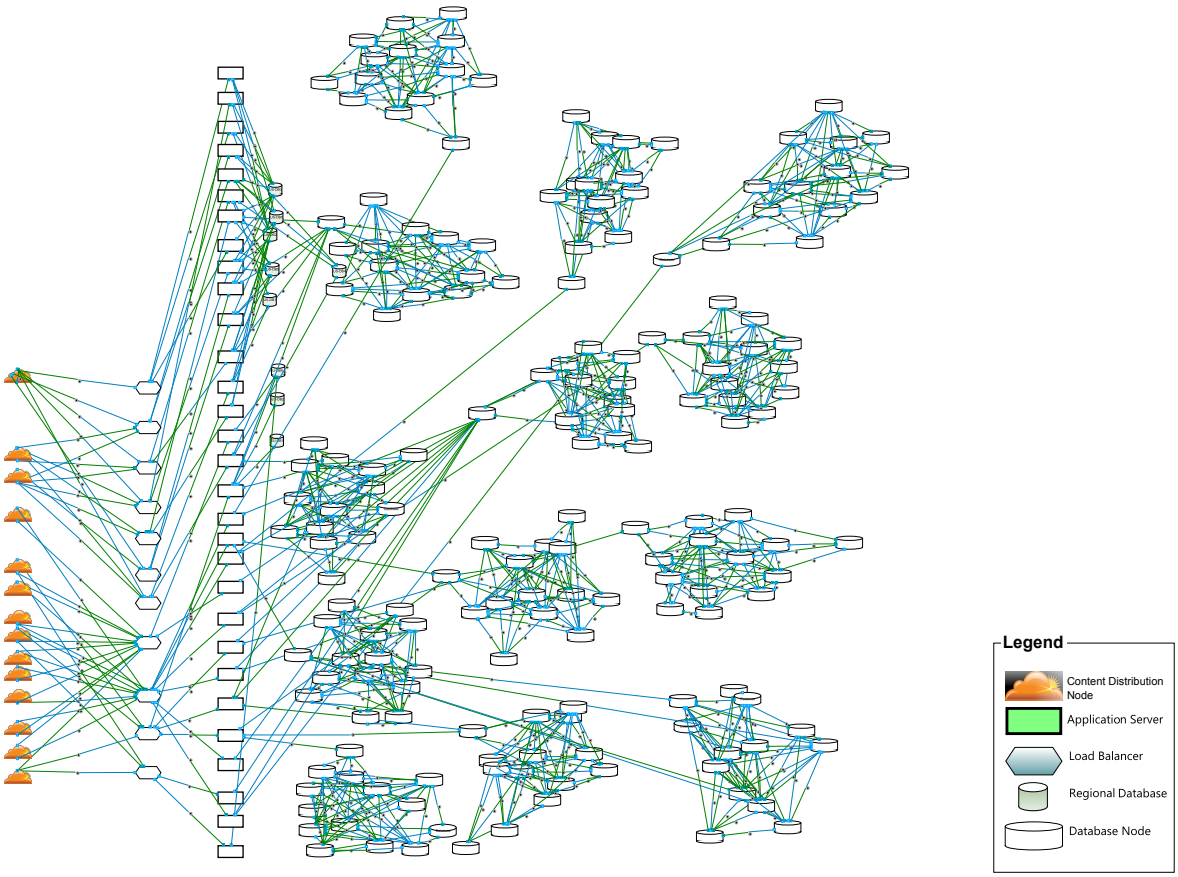



Figure 3: Example architecture of Amazon Web application.

that model-based approaches for clustering sequences are generally better for highly dynamic and large-scale data [8].

We propose a model-based approach to cluster sequences generated by users when interacting with the software architecture graph, and modify it to use to find anomalous sequences [8]. Each user has a role to play in the organization, and we assume that the activities a user carries out during the session is heavily dependent on her role. For instance, an administrator's core set of activities may comprise checking server logs and running special permission programs. However, for a software employee, the main activities will involve logging in to the system, and running programs without any special permissions. Once the role is inferred, a user can choose a sequence of activities from the sequence distribution conditional to that role. This could be written as a generative model, in the following way:

1. A user implicitly chooses what cluster she belongs to. This assignment may depend on her role, or the task that she wants to perform.

2. Given the cluster, then her behavior, in the form of a sequence of activities, is generated from the model from some activity distribution related to that cluster.

As in any model-based approach, it is assumed that data from different users are generated independently. We also assume that a user is equally likely to belong to any cluster. The input to the clustering algorithm is the number of clusters ($K$) and the sequence data. In the algorithm, each cluster is associated with a $n^{th}$-order Markov chain [2]. The probability of an observed sequence $x_0, x_1, x_2...x_d$ belonging to a particular cluster $c_k$ is given by:

$$p(x|c_k) = p(x_0, c_k).\prod_{i=1}^{i=d} p(x_i|x_{i-1}, c_k)$$

where $p(x_0, c_k)$ is the probability of $x_0$ being the starting state of sequences in cluster $c_k$, and $p(x_i|x_{i-1})$ is the transition probability of moving from state $i-1$ to state $i$ in the model specified by cluster $c_k$.

We can assume that each cluster model can be represented by a transition matrix which captures the probability of moving from one state to another in a sequence which belongs to some cluster. These transition matrices are represented by $\theta = (\theta_1, \theta_2, \theta_3....\theta_k)$. Algorithm 1 starts by initializing all the transition matrices randomly, and also initializing sequences into clusters randomly. Then, it keeps on alternating with E-step and M-Step until it reaches convergence. E-Step assumes that transition matrices are fixed, and assigns each sequence to a cluster that maximizes the probability of observing that sequence in the given data. M-Step assumes that cluster assignments are fixed and updates the transition matrix that can best explain the cluster assignments. For experimental purposes, we have set the

[2]The order of the Markov chain can be increased based on scalability and the history of states to include.

convergence condition as one in which there is little change ($10^{-4}$) in the log-likelihood of the data.

**Algorithm 1** Model-based sequence clustering algorithm

**Input:** Number of clusters(K) and sequence data
**Output:** Cluster assignments and scores
Initialize cluster assignments $z_1, z_2, ...z_n$
Initialize transition probability matrices $\theta_1, \theta_2, ...\theta_k$
**while** Not Converged **do**
  **E-Step:** Assign each sequence to most likely cluster
  **for** $i = 0$ to $n$ **do**
    $z_i = \arg\max_{z_i} p(x|c_i)$
  **end for**
  **M-Step:**Update transition matrices $\theta_1, \theta_2, ...\theta_k$
  **for** $z = 1$ to $k$ **do**
    **for** all sequences in cluster $z$ **do**
      Count number of transition edges from $a$ to $b$ in the sequence.
      Update transition matrix $\theta_z$
    **end for**
  **end for**
**end while**

However, the model also requires as input the number of clusters. For large datasets, figuring out the number of clusters can be a problem because sequence clustering is sensitive to this parameter. Different number of clusters as input, could potentially produce very different results, which might not be consistent with what we are trying to capture, or explain the data well. To come up with an appropriate number of clusters, we employ the bayesian information criterion [38] as described in Algorithm box 2.

**Algorithm 2** BIC Algorithm

**Input:** Candidate range of number of clusters ($k_min, k_max$)
**Input:** Number of iterations $iters$
**Output:** Number of clusters
**for** $k = k_min$ to $k_max$ **do**
  **for** $i = 0$ to $iters$ **do**
    Cluster using Algorithm 1.
    Compute log-likelihood $l(D;\theta)$for partition obtained using Algorithm 1 .
    Calculate BIC value = $l(D;\theta) - \frac{k \log(n)}{2}$
  **end for**
  Choose highest BIC value for this cluster $k$
**end for**
Return $k$ corresponding to highest BIC values among all clusters

To find outliers, we take into consideration the probability value with which each sequence may be part of the each cluster. The sequences with high membership scores, which form clusters of their own are candidates for anomalous sequences. We can display these sequences, along with their scores in a sorted fashion. Based on the analyzer's cognition, the top M nodes can be marked as outliers. A sorted list of such outliers with their probability scores can then be presented to the analyst, who can then filter out the false positives, and also take into consideration any changes that might have not been picked up by the system, such as role changes or other information derived from the social context.

After introducing our approach for model-based anomaly detection, we describe in the next section the methodology followed for its evaluation.

# 6. VALIDATION

In this section, we first present our experiment setup, including the approach employed to generate our data set, then we present the results of our clustering approach.

## 6.1 Experiment setup

We model the architecture used for evaluating our approach after the reference architecture employed for Amazon Web application hosting.[3] In the instance employed for our experiments (Figure 3, 242 nodes), we incorporate two geographical areas (U.S. and Europe), each of which includes:

- A *content delivery network* (e.g., Amazon CloudFront) that delivers dynamic content to users and optimizes performance via smart request routing.
- A set of *load balancers* that distribute incoming application traffic among multiple Web application servers.
- A set of *application servers* to process requests that may be deployed, for example, on Amazon EC2 instances.
- *Relational databases* that host application data specific to a particular region.

In addition to these geographical zones, the architecture also includes high-availability *distributed databases* that host additional application data and are shared by the different geographical areas (represented as node clusters on the right-hand side of Figure 3). This architecture style is a classic N-tier Cloud system, of the kind that is used by a number of large organization to store their data and provide scaleable access to many clients.

Unfortunately, we could not have access to a real, large-scale system, and so for validating our approach we needed to generate a large scale software architecture where we could simulate the behavior of a realistic scenario with many users. The first step was to codify the kinds of components and connectors, and rules governing their correct configuration, into an Acme style [17] that could be used to govern the generation of a correct architecture. To generate the architecture instance of an Amazon Web application, we translated this style to Alloy [24], a language based on first-order logic that allows modeling structures (known as *signatures*) and the relations between them as constraints. In particular, we use Alloy to formally specify the set of topological constraints of the architecture, and then use the Alloy analyzer tool to automatically generate models that satisfy those constraints. Listing 1 shows the encoding of the basic signatures for part of the Amazon Web architecture that includes the declaration of content delivery network components, load balancers, and some of the topological constraints that determine how components of these types should be connected among them.

Since Alloy does not scale to find models satisfying the constraints with an arbitrary number of instances of the different signatures, we divided our Alloy specification into two parts that describe different sub-architectures: (i) geographical area, including content delivery networks, load

[3]https://aws.amazon.com/architecture/

```
1  abstract sig comp { conns : set comp } // Abstract component
2  //No component must be connected to itself
3  fact { all n:comp | not n in n.conns }
4  //Content Delivery Network, Load Balancer, and Application Server
5  sig CDN, LB, AS extends comp { }
6  // All components must be reachable at least from one CDN
7  fact { some c:CDN | all n:comp−CDN | n in c.∗conns }
8  //CDNs must be connected only to LBs
9  fact { all c:CDN.conns | c in LB }
10 //Each CDN must be connected at least to some LB
11 fact { all c:CDN | some l:LB | l in c.conns }
12 // LBs must be connected only to AS
13 fact { all l:LB.conns | l in AS }
14 ...
```

Listing 1: Alloy architecture specification excerpt.

balancers, application servers, and databases, and (ii) distributed databases. Next, we ran the Alloy analyzer several times on (i) and (ii) to obtain different instances of the sub-architectures, that we then merged in order to obtain an architecture much larger than what we would be able to construct by directly trying to generate the overall architecture from a monolithic Alloy specification.

This generated architecture is then input to a simulator which creates user traces by following valid connections within the software architecture. Transitions from one node within the architecture to another are chosen randomly with equal probability amongst the valid connections. These transitions are performed after a simulated "think time" that is randomly chosen between 1 and 30 seconds. At each node the trace has a randomly chosen probability, between 1% and 25%, of "returning" (i.e., being the end point of the user trace). The path is then reversed back through the system again with a simulated "think time".

To simulate a compromised system accessing corporate data, an anomaly was manually injected into the set of traces. Specifically, an application server interacting directly with a set of database servers. While this is typical behavior, this is unique because the trace originates with the application servers, something that should not occur given the software architecture. This type of anomaly is representative of a compromised server accessesing corporate data sets to stage the data for future exfiltration.

## 6.2 Results

We apply our sequence-based clustering model to the simulated dataset with the injected anomaly. To find an appropriate number of clusters for our dataset, we run a scan from 2 to 100 using BIC. We found that the appropriate number of clusters was 24. We cluster user sequences into those 24 clusters, obtaining the scores for each of the given sequences. The distribution of these scores is shown in Fig 4. The figure clearly shows that the outlier is to the extreme right of the distribution. This point is the outlier score of the injected anomalous sequence. Again, in Fig 5, we can see that the outlier node is to the extreme right and stands out significantly from the rest. In Fig 6, we present the sequence of activities for some clusters, including the anomalous one. In each of these sub-figures, we represent the graph, including an overlay that represents the sequence of activities as a path on the network. Each node represents a unit in the software architecture, and each edge represents the transition between the consecutive nodes in the sequence. As we mentioned earlier, the organizational roles determine the sequence of activities that a user performs. In our algorithm, the latent clusters formed capture these organizational roles. For instance, cluster12 in our simulation captures the role in which most of the requests are being transferred from both European and US CDN's to the same set of distributed databases. However, the injected anomaly, which forms a cluster of its own, behaves very differently because the trace starts on an interaction with the application server.

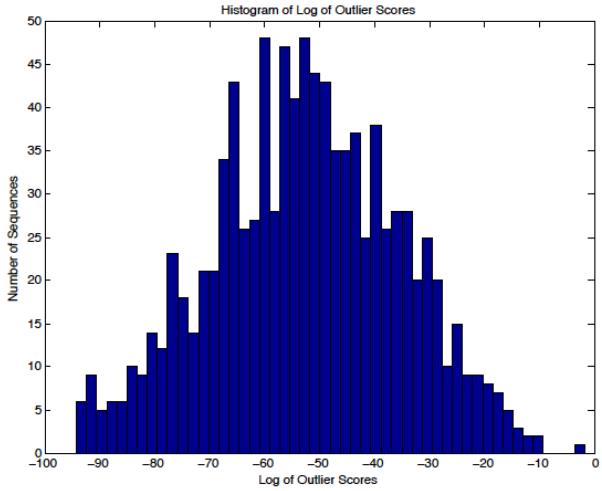



Figure 4: Histogram of the natural logarithm of the Outlier Scores for every sequence generated by the model based sequence clustering algorithm

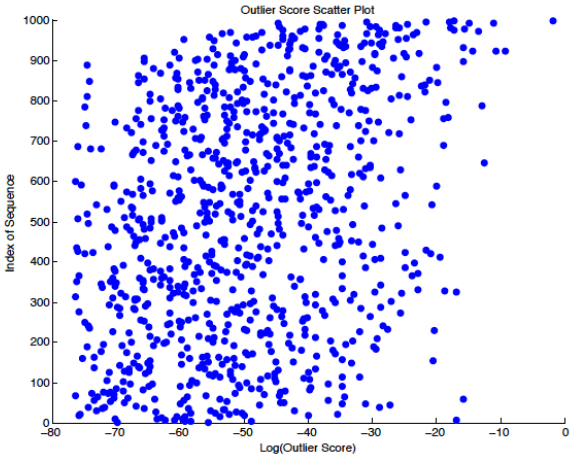
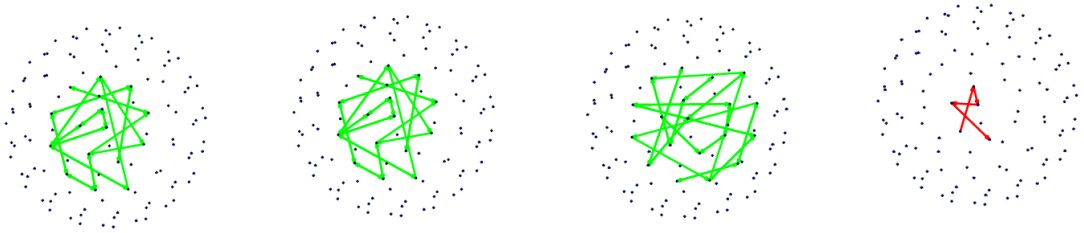



Figure 5: Scatter Plot for the natural logarithm of the outlier score.

# 7. DISCUSSION AND FUTURE WORK

In this paper we have shown the use of a new, model-based clustering algorithm that considers sequences of interaction with components in a software architecture and user roles when clustering. The approach addresses the challenges of applying existing feature- or community-based graph clustering algorithms by considering user interactions with each



(a) Cluster 12 (b) Cluster 22 (c) Cluster 1 (d) Cluster 0

Figure 6: User paths overlaid on the underlying architecture diagram. The first three figures show the combined sequences from all instances that form cluster 12, 22, and 1, respectively. The fourth figure is the anomalous cluster 0.

component in the software architecture, and constructing paths, or traces, that users create when interacting with software. We have shown that our approach can find anomalous user traces in a realistic example of a cloud-based software system. This is the first step in being able to identify insider threats.

While we have demonstrated that the model-based graph clustering algorithm performs well in a realistic setting, there are a number of limitations to our validation that we would like to strengthen in future work. In particular:

- The example system that we used, while based on realistic and reported scenarios, does not use actual real world data. Gaining access to the architectures of real systems, and to actual anomalous data, is extremely difficult. Organizations do not like to make this data public for many reasons, security among them. Our scenario is based on the extensive experience that one of the authors has had in consulting for large international firms that use this kind of infrastructure. However, we would like to address this appeal to expertise with validation on real data, and real systems.

- Because we are using simulated data, it is easy for us to get user traces. In real systems, this is likely to be more challenging. However, we anticipate being able to use standard system instrumentation and complex event processing that we used in [10, 9] to retrieve this information.

- We have made the argument that this approach can find anomalies that other graph clustering approaches cannot, because they do not consider the structure of the system or the role of users in an organization. In future work, we would like to address this by implementing several of these approaches on our data to more strongly verify this claim, and to be able to provide performance and complexity based comparisons between our approach and others.

- While we have shown we can detect an injected anomaly, we have not evaluated the characteristics of the algorithm in terms of its complexity, performance, or precision/recall. The initial results are promising, but we must address these characteristics in order to fully assess our approach.

- We have focused on a particular type of architectural system, large web scale system, and a particular type of anomaly, staging data for exfiltration. While both of these have been constructed to be realistic as possible, future work would include evaluation of different types of applications and different types of security anomalies.

## Acknowledgments

This work is supported in part by the National Security Agency. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Security Agency or the U.S. government.

## 8. REFERENCES

[1] C. C. Aggarwal, Y. Zhao, and P. S. Yu. Outlier detection in graph streams. In *In Proceedings of the 27th International Conference on Data Engineering (ICDE)*, pages 399–409, 2011.

[2] L. Akoglu and C. Faloutsos. Event detection in time series of mobile communication graphs. In *In Proceedings of Army Science Conference*, 2010.

[3] L. Akoglu, M. McGlohon, and C. Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *In Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 410–421, 2010.

[4] L. Akoglu, H. Tong, and D. Koutra. Graph-based anomaly detection and description: A survey. *Data Mining and Knowledge Discovery (DAMI)*, 28(4), 2014.

[5] M. Araujo, S. Papadimitiriou, S. Gunnemann, C. Faloutsos, P. Basu, A. Swami, E. Papalexakis, and D. Koutra. Com2: Fast automatic discovery of temporal(comet) communities. In *In Proceedings of the 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2014.

[6] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos. Netsimilie: A scalable approach to size-independent network similarity. *CoRR,abs/1209.2684*, 2012.

[7] H. Bunke, P. J. Dickinson, A. Humm, C. Irniger, and M. Kraetzl. Computer network monitoring and abnormal event detection using graph matching and multidimensional scaling. In *In Proceedings of 6th Industrial Conference on Data Mining (ICDM)*, pages 576–590, 2006.

[8] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Visualization of navigation patterns on a web site using model-based clustering. In *Proceedings of ACM SIGKDD*, pages 280–284, 2000.

[9] P. Casanova, D. Garlan, B. Schmerl, and R. Abreu. Diagnosing architectural run-time failures. In *In Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2013.

[10] P. Casanova, B. Schmerl, D. Garlan, and R. Abreu. Architecture-based run-time fault diagnosis. In *In Proceedings of the 5th European Conference on Software Architecture*, 2011.

[11] D. Chakrabarti. Autopart: parameter-free graph partitioning and outlier detection. In *In Proceedings of the 8th European Conference on Principles and Practices of Knowledge Discovery in Databases*, pages 112–124, 2004.

[12] A. Cummings, T. Lewellen, D. McIntire, A. Moore, and R. Trzeciak. Insider threat study:illicit cyber activity involving fraud in the US financial services sector. *Special Report, CERT, Software Engineering Institute*, 2012.

[13] P. Drineas, R. Kannan, and M. W. Mahoney. Fast monte carlo algorithms for matrices: Computing a compressed approximate matrix decomposition. *SIAM Journal on Computing*, 36:184–206, 2006.

[14] D. B. et. al. Stinger:spatio-temporal interaction networks and graphs extensible representation. In *Technical Report*, 2009.

[15] Y. Fu, K. Sandhu, and M.-Y. Shih. Clustering of web users based on access patterns. *Web Usage Analysis and User Profiling*, pages 21–38, 2000.

[16] J. Gao, F. Liang, W. Fan, C. Fang, Y. Sun, and J. Han. On community outliers and their efficient detection in information networks. In *In Proceedings of the 16th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 813–822, 2010.

[17] D. Garlan, R. T. Monroe, and D. Wile. Acme: Architectural description of component-based systems. In *Foundations of Component-Based Systems, KEY = garlan, PAGES = 47-68, EDITOR = Leavens, Gary T. and Sitaraman, Murali, PUBLISHER = Cambridge University Press*. 2000.

[18] M. E. Gaston, M. Kraetzl, and W. D. Wallis. Using graph diameter for change detection in dynamic networks. In *In Australian Journal of Combinatorics*, pages 299–311, 2006.

[19] M. Gupta, J. Gao, Y. Sun, and J. Han. Integrating community matching and outlier detection for mining evolutionary community outliers. In *In Proceedings of the 18th ACM International Conference on Knowledge Discovery and Data Mining*, pages 859–867, 2012.

[20] D. Hawkins. Identification of outliers. *Chapman and Hill*, 1980.

[21] N. A. Heard, D. J. Weston, K. Platanioti, and D. J. Hand. Bayesian anomaly detection methods for social networks. *Annals of Applied Statistics*, 4:645–662, 2010.

[22] K. Henderson, B. Gallagher, L. Li, L. Akoglu, T. Eliassi-Rad, H. Tong, and C. Faloutsos. It's who you know: graph mining using recursive structural features. In *In Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining(SIGKDD)*, pages 663–671, 2011.

[23] T. Ide and H. Kashima. Eigenspace-based anomaly detection in computer systems. In *In Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 440–449, 2004.

[24] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[25] R. Johnson, editor.

[26] M. Keeney, D. Capelli, E. Kowalski, A. Moore, T. Shimeall, and S. Rogers. Insider threat study: Computer sabotage in critical infrastructure sectors. In *CERT Program and Software Engineering Institute*, 2005.

[27] D. Koutra, E. Papalexakis, and C. Faloutsos. Tensorplat:spotting latent anomalies in time. In *In 16th Panhellenic Conference on Informatics (PCI)*, 2012.

[28] D. Koutra, J. Vogelstein, and C. Faloutsos. Deltacon: A principled massive-graph similarity function. In *In Proceedings of the 13th SIAM International Conference on Data Mining (SDM)*, 2013.

[29] E. Kowalski, D. Cappelli, and A. Moore. Insider threat study: Illicit cyber activity in the government sector. *Software Engineering Institute*, 2008.

[30] E. Kowalski, D. Cappelli, and A. Moore. Insider threat study: Illicit cyber activity in the information technology and telecommunications sector. *Software Engineering Institute*, 2008.

[31] C. Liu, X. Yan, H. Yo, J. Han, and P. S. Yu. Mining behavior graphs for "backtrace" of non crashing bugs. In *In Proceedings of the 5th SIAM International Conference on Data Mining (SDM)*, 2005.

[32] E. Muller, P. Iglesias, Y. Muller, and B. Klemens. Ranking outlier nodes in subspaces of attributed graphs. In *In Proceedings of the 4th International Workshop on Graph Data Management: Techniques and Applications*, 2013.

[33] C. C. Noble and D. J. Cook. Graph-based anomaly detection. In *In Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 631–636, 2003.

[34] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina. Web graph similarity for anomaly detection. *Journal of International Services and Applications*, 1:1167, 2008.

[35] B. Perozzi, L. Akoglu, P. Iglesias, and E. Muller. Focused clustering and outlier detection in large attributed graphs. In *In ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD)*, 2014.

[36] B. Pincombe. Anomaly detection in time series of graphs using arma processes. *ASOR Bulletin*, 2005.

[37] R. A. Rossi, B. Gallagher, J. Neville, and K. Henderson. Role-dynamics: fast mining of large dynamic networks. In *In Proceedings of the 21st International Conference on World Wide Web (WWW)*, pages 997–1006, 2012.

[38] G. E. Schwartz. Estimating the dimnesson of a model. *Annals of Statistics*, 6:461–464, 1978.

[39] T. E. Senator, H. G. Goldberg, A. Memory, W. T. Young, B. Rees, R. Pierce, D. Huang, M. Reardon, D. A. Bader, E. Chow, I. A. Essa, J. Jones, V. Bettadapura, D. H. Chau, O. Green, O. Kaya, A. Zakrzewska, E. Briscoe, R. L. M. IV, R. McColl, L. Weiss, T. G. Dietterich, A. Fern, W. Wong, S. Das, A. Emmott, J. Irvine, J. Y. Lee, D. Koutra, C. Faloutsos, D. D. Corkill, L. Friedland, A. Gentzel, and D. Jensen. Detecting insider threats in a real corporate database of computer usage activity. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 1393–1401, 2013.

[40] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[41] P. Shoubridge, M. Kraetzl, W. D. Wallis, and H. Bunke. Detection of abnormal change in dynamic networks. *Journal of Interconnection Networks*, 3:85–101, 2002.

[42] H. Sun, J. Huang, J. Han, H. Deng, P. Zhao, and B. Feng. Gskeletonclu:density based network clustering via structure connected tree division or agglomeration. In *In Proceedings of the 10th IEEE International Conference on Data Mining (ICDM)*, pages 481–490, 2010.

[43] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. Graph-scope: parameter free mining of large time-evolving graphs. In *In Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining*, pages 687–696, 2007.

[44] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *In Proceedings of the 5th IEEE International Conference on Data Mining (ICDM)*, pages 418–425, 2005.

[45] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *In Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 374–383, 2006.

[46] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos. Less is more: Sparse graph mining with compact matrix decomposition. *Statistical Analysis and Data Mining*, 1:6–22, 2008.

[47] H. Tong and C.-Y. Lin. Non-negative residual matrix factorization with application to graph anomaly detection. In *In Proceedings of the 11th SIAM International Conference on Data Mining(SDM)*, pages 143–153, 2011.

[48] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger. Scan: a structural clustering algorithm for networks. In *In Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 824–833, 2007.