GraphBAD: A General Technique for Anomaly Detection in Security Information and Event Management

Simon Parkinson[1], Mauro Vallati[1], Andrew Crampton[1], and Shirin Sohrabi[2]

[1]*Department of Computer Science, University of Huddersfield, UK*
[2]*IBM T.J Watson Research Center, New York, USA*

Abstract

The reliance on expert knowledge –required for analysing security logs and performing security audits– has created an unhealthy balance where many computer users are not able to correctly audit their security configurations and react to potential security threats. The decreasing cost of IT and the increasing use of technology in domestic life is exacerbating this problem where small companies and home IT users are not able to afford the price of experts for auditing their systems configuration.

In this paper we present GraphBAD, a graph-based analysis tool able to analyse security configurations in order to identify anomalies that could lead to potential security risks. GraphBAD, which does not require any prior domain knowledge, generates graph-based models from security configuration data and, by analysing such models, is able to propose mitigation plans that can help computer users in increasing the security of their systems. A large experimental analysis, conducted on both publicly available (the well-known KDD dataset) and synthetically generated testing sets (file system permissions), demonstrates the ability of GraphBAD in correctly identifying security configurations anomalies and suggesting appropriate mitigation plans.

Key words: Security Auditing, Log Files, Graph Structure, Anomaly Detection, SIEM

1 Introduction

Auditing security configurations is the process of searching for anomalies that potentially expose a vulnerability of a considered system. The term Security Information and Event Management (SIEM) is often used to describe this process of monitoring audit logs and security configurations to identify vulnerabilities. Given the complexity of the task, there is a heavy reliance on expert knowledge, which is required for understanding the different security configurations. This reliance has two main drawbacks: first, expert knowledge can be incomplete or erroneous; second, the high cost of security experts makes it infeasible for many users to correctly configure the security of their Information

Technology (IT) systems. Therefore, in many cases unseen weaknesses, which can be exploited, will remain in the configuration. Clearly, auditing security configurations is a critical problem for organisations whom significantly rely on their IT infrastructure for undertaking business. For this reason, many companies frequently employ a third party security auditing company to examine their IT infrastructure to identify weaknesses and suggest suitable mitigation plans (named *penetration testing* [1, 2]). Although businesses are prepared to pay a premium for maintaining their security, the average home IT user or small company are left to maintain their own IT security. Furthermore, the decreasing cost of IT and the increasing use of technology in domestic life is exacerbating this problem.

These limitations are heightened by the following factors: (i) The complexity of computational infrastructure –the cyber-physical platform on which security is attained– is increasing and the technological landscape is dynamic; (ii) Vast amounts of data available at any given time present a signal-to-noise ratio that is impossible to manage without technological advanced support [3]; (iii) The diversity, completeness and spatio-temporal distribution of security data across different sources compound the difficulty of extracting important signals from an ocean of noise [4]; (iv) Security threats, knowledge capabilities of human security experts, and technological support rarely co-evolve in such a fast changing discipline, limiting the ability to defend against new threats appropriately.

In order to overcome these factors, it is envisaged that the design and development of automatic approaches, able to identify security anomalies and suggest actions to be taken, will assist in mitigating security risks. The underlying need to produce *situation aware* tools is widely acknowledged, and actions are being taken at state level [5]. Various studies have demonstrated how software tools can be developed to extract meaningful knowledge to aid security configuration, auditing, and digital investigations [6]. Such tools are domain-dependent in that their functionality is heuristically guided to identify threats that are expected. A limitation of these tools is that each requires different knowledge and skills to translate their output to gain an appreciation of why the extracted knowledge is of significance [7], i.e. to autonomously gain situation awareness which can be exploited to promote cyber resilient systems.

In this paper we propose a Graph-based Security Anomaly Detection herein named GraphBAD. GraphBAD analyses security configurations and audit logs in order to identify anomalies which can potentially lead to security issues. The analysis is performed by encoding the provided input, which describes the configuration of a considered IT system, under the form of an undirected graph, and by checking the regularity of subgraphs. By analysing the models generated from the data, GraphBAD identifies anomalies and then suggests suitable mitigation plans that enable users to take reactive action; requiring minimal expertise from the final user. Evidently, the proposed approach can be used by non-experts, but it can also be fruitfully exploited by experts either as a decision support or training tool. In the former use, it can help the expert by quickly analysing large chunks of data; in the latter it can be used for expert training. Remarkably, the proposed system performs anomaly detection by analysing the generated graphs. Therefore, it is applicable to a wide range of scenarios, in which irregularities can be spotted by analysing undirected graphs.

To empirically evaluate the effectiveness of the proposed approach, a large experimental analysis has been performed. Such analysis considered large synthetically generated file system permission datasets, as well as the well-known (KDD Cup 1999 datasets) [8], which has been used in the Third International Knowledge Discovery and Data Mining Tools Competition. There are many other datasets available in the public domain for benchmarking of anomaly detection algorithms; however, the KDD dataset is chosen for its applicability to the SIEM theme of this research and many other researchers have benchmarked their algorithms against the dataset, allowing for direct comparisons to be made. Our results indicate that GraphBAD is an effective approach for identifying anomalies in security configurations and suggesting suitable mitigation plans.

The primary aim of this paper is to introduce a general and widely exploitable technique for identifying anomalies in SIEM data sources. The research hypothesis addressed in this paper is: representing security data using graph-based structure creates the potential to identify anomalies without programatically encoding specific application knowledge. The primary contributions presented in this paper are:

- **GraphBAD** for the unsupervised translation of structured data into a graph representation that can then be analysed in order to identify anomalies. A measure of regularity is then used to identify anomalous subgraphs. This technique, albeit an exploratory prototype, demonstrates the potential of adopting the domain-independent philosophy for application in the security auditing and SIEM communities. Furthermore, the application will be of importance in the Artificial Intelligence community as a useful benchmark for further development and evaluation of unsupervised learning mechanism.

- **Method of generating synthetic datasets** has been developed in response to the absence of publicly available benchmarking security configuration datasets. The technique is used for generating synthetic datasets for file system access control. The auditing of file system permissions is a prominent challenge in the security auditing community as vulnerabilities can have significant implications [9, 10]. The technique will promote awareness and result in other researchers using the datasets for benchmarking purposes, thus providing new solutions for file system auditing.

- **Benchmarking** using large security logs to identify anomalies (KDD dataset). An accuracy rate of 0.80 demonstrates the potential of GraphBAD as the result surpasses some supervised approaches. Furthermore, to the best of the authors knowledge, the method of partitioned KDD data presented in this paper aids to remove potential bias introduced in previous research where the data has been selectively divided into small subsets and only utilised a few attack types.

- **Worked example for detecting anomalies in file system permissions** allows the reader to relate the proposed technique to a common administrative and audit challenge. This ensures that the research presented in this paper has importance for both the research and administrative communities.

The paper is organised as follows: as graph based systems are used in many different applications within the cyber security research discipline, the next section is devoted to summarising the different areas of research and the positioning of this paper. The next section is devoted to a detailed explanation and example of the presented graph-based extraction technique. This leads to the experimental analysis section where a technique is presented to detect anomalous data items in both synthetically generated and bench marking datasets. Following this, conclusions and the direction of further work is provided.

2 Related Works

In terms of research in the Security Information and Event Management (SIEM) and administrative communities, there is a wealth of research detailing the challenges of analysing data sources to assist users in auditing their systems' security. Recent review papers detail the challenges facing SIEM analysis, and in particular in the areas of intrusion [11] and attack detection [12]. Research has demonstrated the potential of using data mining techniques to identify hidden patterns in malware data to support SIEM analysis [13]. Another interesting approach is the use of latent semantic analysis to help reduce unnecessary noise in large datasets [14]. These approach demonstrates good potential; however, knowledge regarding data structure, as well as what constitutes a threat, is required throughout the analysis. This motivates the work in this paper where an unsupervised techniques is developed to assist in SIEM. According to literature, this view is shared by the SIEM research community where although the use of statistical and machine learning techniques have demonstrated some success, yet they require modification per problem instance [15].

Graph-based analysis is used in many different aspects of cyber security [16] which warrants its exploitative use in this research. Three main strands of research exploiting graph-based models can be identified as:

1. Analysis of software behaviour. Works in this area are encoding actions executed by software as graphs, in order to identify suspicious patterns, typical of malicious code [17]. For instance, recently graph-based analysis has been used for identifying malicious downloader systems [18].

2. Penetration testing / attack graphs. This area focuses on identifying sequences of actions that violates some property defined over the system. Researchers have successfully used graph-based representations to identify vulnerabilities in networks [19, 20, 21, 22]. In addition, researchers have also developed methods of automating penetration testing and auditing [23, 24]. The interested reader is referred to [16] for a survey on this topic.

3. Identification of anomalies or potential issues in security configuration. Notably, a number of works investigated the extraction of graph-based models from security data (see, e.g. [25, 26, 27, 28, 29, 30, 9]). However, most of them are aimed at

identifying anomalies in a graph's structure with prior expert knowledge regarding the data's structure. The interested reader is referred to [31] for an overview of the exploitation of graph-based anomaly detection techniques, in a wide range of application.

According to the provided "classification", our work should be situated in the third category. However, it does also have practical applications in the second category where configuration files and logs are analysed during penetration testing. Despite the amount of existing work on this topic, there is still a significant lack of research being performed in the area of developing general, unsupervised, autonomous methods of anomaly detection that can provide mitigation suggestions without relying on expert knowledge. There is a large body of research work on using graph-based models in cyber security analysis, and some of the more prominent have been mentioned in the above. In addition, there are works where automated planning and scheduling are used for security analysis [23, 32]. However, the majority of this work is "domain-specific" in that specific application knowledge is encoded into the algorithms or provided by the user as a hand-crafted model. Although such work presents significant findings in its related application area, it is unable to provide contribution to the research hypothesis integral to this paper.

There are different types of anomalies within the research area of SIEM cyber security analysis, which can loosely be categorised as:

1. **Temporal** anomalies occur when data is determined to be irregular if its temporal occurrence is unusual. For example, identifying temporal anomalies in internet traffic [33], and techniques detecting temporal based anomalies in data sources available on UNIX systems.

2. **Volume** anomalies occur where a spike in the volume of data being monitored and/or logged occurs. For example, detecting volumetric changes in network traffic [34, 35], which can be used for security applications such as detecting a high volume of command and control communication, indicative of a malware [36, 37].

3. **Relational** where an anomaly is described by an irregular relationship among data elements (objects). There has been a wealth of research in developing and exploiting new techniques for detecting security configuration anomalies, which adopt an object-centric modelling technique [38, 39, 40, 41]

Based on the above categorisation, this paper focusses on the development of techniques for the identification of relational anomalies. It is worth noting that in focusing on the development of a technique capable of the unsupervised identification of irregularities in relational data, the potential applicability of the proposed technique on other types of anomalies will be hindered. This is due the different characteristics with datasets in the volume and temporal categorise. For example, temporal datasets will have to deal with continuous (e.g. time), and datasets of a large volume will pose computation and efficiency challenges.

3 THE GraphBAD APPROACH

Before introducing the proposed Graph-based Security Anomaly Detection (GraphBAD) approach, we provide a definition of *anomaly*, which follows from the definition provided in [31].

Definition 1 *Given a graph and a measure of regularity, an anomaly is a subgraph that has a regularity value below the provided reference measure.*

Notably, given the generality of the proposed approach, anomalies can be identified only by considering the structure of the analysed graph. Having introduced the notion of anomalies, we can now present GraphBAD. An overview of the system is as follows:

1. **Graph-based model generation** for the translation and reduction of security auditing data into a graph-based representation;
2. **Anomalous subgraphs detection** where data entries are determined to be anomalous should they be infrequent and poorly connected;
3. **Mitigation suggestion generation** for suggesting the objects that need to be reducing to nullify the identified vulnerability.

Each step will be described in the remainder of this section. As a working example to aid the reader (researcher and system administrator), throughout the paper we consider auditing access control on Microsoft's New Technology File System (NTFS). The brief construct of access control in NTFS (sufficient to understand the example) is that each user or group (e.g. *Bob*) within the system can be granted a level of access (e.g. $FileReadData$) on a file system resource ($k : \backslash$) through the use of fine-grained permission attributes. Users can be associated to groups which inherit multiple access control entries on multiple file system resources. This combination, amongst others not covered here, make administering and auditing NTFS permissions challenging and thus the solution presented in this paper is of significant benefit to this target domain. The interested reader is referred to [42] for a full description of NTFS permissions and their auditing challenges. The use of this example will communicate how GraphBAD is able to work domain-independently.

3.1 Graph-based Model Generation

The first step of the approach is to convert the provided security data into a relational graph-based model. Here the model is represented as an undirected graph. As security configuration data and audit logs comes in many different forms, the developed algorithm relies on two assumptions: (i) security data are provided in text form; and (ii) each delimited value represents an object and all objects in the same dataset share a relationship. It should be noted that currently, many security configurations respect the mentioned assumptions [11]. Furthermore it should be mentioned that other researchers also acknowledge this assumption. This is most likely due to the natural way in which a programmer will rationalise security configurations following an *object-centric* philosophy.

For each object identified in the security data, a corresponding vertex v_k is generated. Therefore, given n objects, a set of vertices $V = \{v_1, v_2, ..., v_n\}$ is provided. Following the file system example, reading the two datasets of:

$$\begin{aligned} Bob, k:\backslash, FileReadData \\ Dave, k:\backslash, FileReadData \end{aligned}$$

would result in the following vertices:

$$V = \{Bob, Dave, k:\backslash, FileReadData\}$$

If an object –corresponding to a vertex v_k in set V– is part of a dataset containing more objects, than for each couple of vertices v_k,v_i with $k \neq i$, an edge $e_i = (v_k, v_i)$ is created. At the end of this step, an edge set $E = \{e_1, e_2, ..., e_j\}$ is generated. Following our example, the considered datasets would result in the production of the following relationship set.

$$\begin{aligned} e_1 &= (Bob, k:\backslash) \\ e_2 &= (Bob, FileReadData) \\ e_3 &= (k:\backslash, FileReadData) \\ e_4 &= (Dave, k:\backslash) \\ e_5 &= (Dave, FileReadData) \\ e_6 &= (k:\backslash, FileReadData) \end{aligned}$$

3.1.1 Graph Reduction

Once all datasets have been processed, the cardinality of both V and E are likely to be excessively large and contain a significant quantity of repetitive information. Intuitively, it is possible to reduce the size of the graph without losing information, in order to speed up the subsequent analysis. Performing reduction has potential to group regular data entries together, making irregular items less distinguishable. Conversely, it is also possible that irregular data entries will be grouped together making them more distinguishable. This does create uncertainty over its potential. However, empirical analysis performed in this paper yields positive results. Vertices that are not directly connected, but share the same set of edges to other vertices, are merged. In other words, vertices' relationships determine their class (set). In the example, it is easy to determine that *Bob* and *Dave* can be placed into the same set as they have the same relationships with the same objects. In order to perform this reduction, each vertex is systematically compared with all other vertices. A given vertex v_k is determined to match v_i where $k \neq i$ if $e_i = e_k$. When a match is found, both v_k and v_i will be added to a new or previously created set. For example, $s_1 = \{v_k, v_i\}$. This exhaustive search has a complexity of $O(n^2)$. To

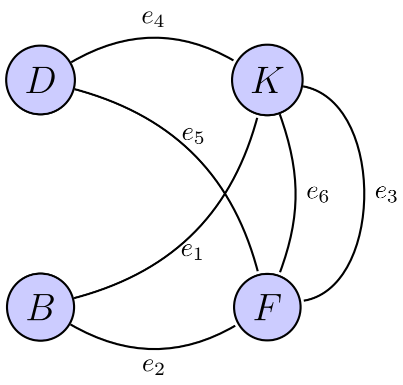

Figure 1: Before graph-based diagram generated by GraphBAD for the considered file system example. $D = Dave$, $B = Bob$, $K = k : \backslash$, and $F = FileReadData$

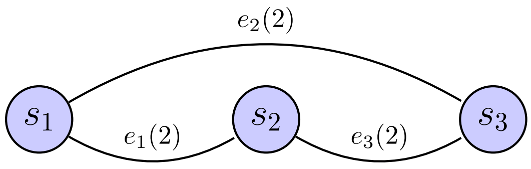

Figure 2: Final graph-based diagram generated by GraphBAD for the considered file system example. Cardinality (weight) of edges is provided between brackets.

aid understanding, consider the following continuation to the file system example where the following sets are determined:

$$\begin{aligned} s_1 &= \{Bob, Dave\} \\ s_2 &= \{k : \backslash\} \\ s_3 &= \{FileReadData\} \end{aligned}$$

Relationship reduction is performed to remove repetitive edges. In the example, e_6 is removed as $e_6 = e_3$. To each "surviving" edge we attached the number of edges removed because of it: this information gives the weight (or cardinality) of each edge in the graph.

3.1.2 Rename Vertices and Perform Final Edge Reduction

In the final step, the graph is re-generated by collapsing vertices of the same set into a single vertex and removing duplicate edges, generated by vertices reduction. In our example, e_1 and e_4 are now equal and one of them can safely be removed. The before and final graph $G = \{\{s_1, s_2, s_3\}, \{e_1(2), e_2(2), e_3(2)\}\}$ of the considered example are shown in Figure 1 and Figure 2, respectively.

$$
\begin{aligned}
&& e_1 &= (s_1, s_2)\\
s_1 &= \{User_1, ..., User_{20}\} & e_2 &= (s_1, s_3)\\
s_2 &= \{k : \backslash\} & e_3 &= (s_2, s_3)\\
s_3 &= \{FileReadData\} & e_4 &= (s_2, s_4)\\
s_4 &= \{User_{21}\} & e_5 &= (s_2, s_5)\\
s_5 &= \{FileWriteData\} & e_6 &= (s_3, s_4)\\
&& e_7 &= (s_4, s_5)
\end{aligned}
$$

Figure 3: Vertices s_j and edges e_i of the graph generated for the file system anomaly example.

3.2 Anomalous Subgraphs Detection

The graph generated by following the process described in Section 3.1 is examined in order to identify anomalies that can indicate security issues. Here we will consider the continuation of the previous file system example. However, this time there are twenty-one users in the system, all with the permission of:

$$User_n, k : \backslash, FileReadData$$

In addition to each user having the permission of $FileReadData$, $User_{21}$ also has the permission entry of:

$$User_{21}, k : \backslash, FileWriteData$$

Figure 3 provides the vertices and edges obtained by generating the graph. The corresponding graph-based diagram is shown in Figure 4. For the purposes of this example, this extra permission for $User_{21}$ is in fact the anomaly.

The method developed for the automatic detection of anomalous subgraphs is based on an evaluation of weights of edges connecting subgraphs. It should be noted that similar evaluations have been exploited in different domains –for instance in detecting fraud in healthcare data [43]. However, such techniques are either based on supervised learning, or are developed with embedded application knowledge. In GraphBAD, anomalous subgraphs are detected without relying on any prior or specific application knowledge.

3.2.1 Calculate Measure of Regularity

For detecting anomalous subgraphs, also following Definition 1, it is critical to identify a measure of regularity. For this purpose, we consider the average weight of edges per vertex, $R(G)$, calculated as:

$$R(G) = \frac{1}{l} \sum_{k=1}^{j} weight(e_k), \tag{1}$$

where j is the number of edges and l is the number of vertices of the graph G. In the considered example, $R(G) = 17.4$. Although this measure of regularity may appear to be simplistic, it is important to remember that GraphBAD is being used to identify subgraphs that appear anomalous, i.e subgraphs which are below a measure of regularity. Intuitively, the average weight of vertex edges will provide an approximate measure of regularity. More sophisticated and complex measures of regularity can easily be added to GraphBAD; however, the experimental analysis provided in Section 4 does not suggest this to be necessary.

3.2.2 List Suspicious Subgraphs

Once the $R(G)$ value is calculated, it is used as a threshold for identifying interesting and potentially anomalous subgraphs. Specifically, in this context, we are interested in subgraphs g_i with $H(g_i)$ value lower than $R(G)$. Given a subgraph g_i, $H(g_i)$ is the sum of all weighted edges connected to the vertices in g_i. In our example, given $g_1 = \{\{s_1\}, \{\emptyset\}\}$, $H(g_1) = 40$.

Algorithm 1 is then performed for identifying interesting subgraphs. The complexity of the algorithm is $O(n^2)$ and takes three parameters: the threshold value T –corresponding to $R(G)$–, the set of vertices V and the set of edges E. Firstly, the set of interesting subgraphs *belowT* is empty, and the set of subgraphs to compute is initialised with V. This corresponds to considering all the subgraphs of cardinality 1, with no edges. Lines 5–14 evaluate the $H(g_i)$ value of each subgraph: subgraphs with a value above the threshold are removed, while the others are saved in the *belowT* set. At line 10 the cardinality of subgraphs is checked, in order to stop the evaluation as soon as the maximum size –i.e., the size of the graph G– is reached in the next step. At line 16, each subgraph that is still relevant is extended. More specifically, a new connected vertex is added as the aim is to find the largest subgraph with a $H(g_i)$ value less or equal to $R(G)$. Algorithm 2 details the ExtendSubG procedure. As previously mentioned, the purpose of this procedure is to extend a given subgraph, g, by one vertex. Therefore, it is increasing the size of g to find the largest expansion of a subgraph below the threshold. This technique iterates over the set of edges and vertices to identify a vertex, v_k, which is connected to a vertex g_j within the subgraph g by edge e_i. If a suitable vertex is identified, then it and the connected edge is added to g. Finally, the list of all subgraphs g_i with $H(g_i)$ value below threshold is returned. In our example, the returned *belowT* includes three subgraphs: $g_1 = \{\{s_4\}, \{\emptyset\}\}$, $g_2 = \{\{s_5\}, \{\emptyset\}\}$, $g_3 = \{\{s_4, s_5\}, \{e_7\}\}$. Note that in a very regular graph-based model of security data, the described method would return only subgraphs composed of at most a single vertex. Therefore, the number and sizes of subgraphs with H value below the threshold is itself an indication of the regularity of the analysed security data.

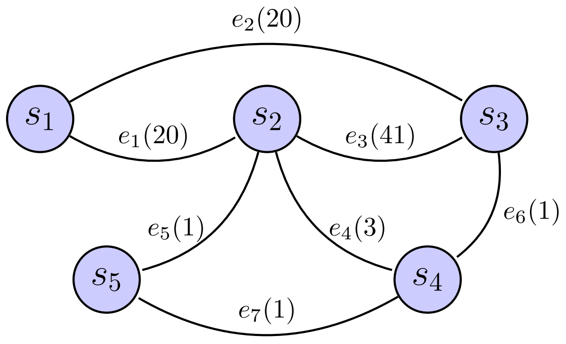

Figure 4: Graph-based diagram of the anomaly example considered for the anomalous subgraphs detection.

3.2.3 Identify Anomalous Subgraphs

In order to provide effective mitigation suggestions, it is important to identify the largest areas of the graph-model G that are potentially affected by the security issue. Therefore, we select the largest subgraphs –with regards to involved vertices– with H value below the threshold. In our example, g_3 is the largest subgraph, involving s_4 or s_5, and is thus identified as anomalous. Again we note that in large datasets there can be more than one security issue, leading to more than one anomalous subgraph.

It is important to identify the maximum number of vertices which are anomalous. For this reason, we select the largest subgraph that involves vertices appearing in other smaller identified anomalous subgraph. In the example this would result in the identification of the following subgraphs: $H_1 = \{s_4\}$, $H_2 = \{s_5\}$, and $H_3 = \{s_4, s_5\}$ with a $m(H_n)$ score of 5, 2, and 7, respectively. If the subgraph of H_3 is below the threshold, then smaller graphs consisting of its constituent vertices (s_4, s_5) are also going to be below the threshold. The solution is to select the subgraph with the greater $m(H_n)$ score under the threshold. Using this technique will disregard H_1, H_2 and result in the identification of anomaly $H_3 = \{User_2 1, FileWriteData\}$.

3.3 Generate Mitigation Suggestions

Given an identified anomalous subgraph, GraphBAD suggests corrective steps in order to remove the anomaly. However, it is useful to remind the reader that no prior knowledge is provided, and that it is pivotal to minimise disruption to the system and its users.

Evidently, modifications can in principle imply the removal of both vertices and edges. This raises the interesting question of will removing the edge, vertex, or both mitigate the anomaly and leave the system in a safe state? It is challenging to discover the answer to such a question without domain knowledge, or the ability to interact with the system to test mitigation hypothesis. However, the proposed technique only considers the removal of vertices as although removing an edge would disable the anomalous subgraph, there is a large amount of uncertainty surrounding the potential effect of the remaining vertices

Algorithm 1 Return all g_i with $H(g_i) < R(G)$

```
1: procedure AnomalousSubgraph($T$,$E$,$V$)
2:     $belowT := \emptyset$
3:     $toCompute := V$
4:     repeat
5:         for g $\in toCompute$ do
6:             if H(g) $\geq T$ then
7:                 $toCompute := toCompute \setminus$ g
8:             else
9:                 $belowT := belowT \cup$ g
10:                if NumberVert(g) = ($|V|$ -1 ) then
11:                    $toCompute := toCompute \setminus$ g
12:                end if
13:            end if
14:        end for
15:        for g $\in toCompute$ do
16:            g := ExtendSubG(g,$E$,$V$)
17:        end for
18:    until $ToCompute \neq \emptyset$
19:    return $belowT$
20: end procedure
```

in the real-world implementation of the data being analysed.

Following in the same object-centric assumption made during the graph generation phase can help identify the solution. For example, removing the vertex in a file system anomaly will also nullify the edge. Domain knowledge allows us to know this; however, in the situation where this assumption is void, it is safer to remove objects, rather than removing their relationships only. For example, removing an edge would result in the anomalous object remaining active in the system, and potentially remaining configured in other anomalies within the system.

Continuing with the illustrative example (Figure 4), subgraph g_3 has been identified as an anomaly. Focusing on vertices, g_3 includes two of them: $\{User_{21}, FileWriteData\}$ (here we refer to the objects which vertices represent, since it makes easier the explanation). Those familiar with file system access control will correctly identify that removing $FileWriteData$ for $User_{21}$ will remove the anomaly. However, this may not be so obvious for users who have little or no prior knowledge regarding file system access control.

Our approach tackles this problem by considering the degree of connectivity that each vertex has to vertices outside the anomalous subgraph. The ideal solution is to be able to remove sets of vertices from the subgraph which have no connectivity beyond the subgraphs; however, in most cases it will not be such a trivial solution. In the example, neither s_4 nor s_5 can be removed under the assumption that they have no connectivity outside of the anomalous subgraph g_3. In cases like the one considered in our example,

Algorithm 2 ExtendSubG for expanding a subgraph, **g**, by one vertex

```
1: procedure ExtendSubG($g$,$E$,$V$)
2:     $i = |E|$, $j = |g|$, $k = |V|$,
3:     for $e_i \in E$ do
4:         for $g_j \in g$ do
5:             if $g_j \in e_i$ then
6:                 for $v_k \in V$ do
7:                     if $v_k \in e_i$ then
8:                         return $\mathbf{g} \cup \{e_i, v_k\}$
9:                     end if
10:                end for
11:            end if
12:        end for
13:    end for
14:    return $null$
15: end procedure
```

the vertex with the lowest connectivity value is removed. Here, as connectivity, we consider the sum of the weights of edges connecting the vertex with vertices outside the anomalous subgraph. In the example, this would result in $s_4 = 8$, $(e_4 + e_6)$ and $s_5 = 2$, $(s_6 = 2)$. As s_5 has the lowest score, it is considered as the vertex which should be removed.

Once a vertex has been identified to be removed, this information is presented to the user. Specifically, positive (i.e., removing the anomaly) and negative (i.e., affecting relationships outside the anomalous subgraph) effects of removing the object (or more objects) corresponding to the vertex in the graph model are described. In the example, the following is presented to the user:

$$\begin{aligned}
&Remove\ s_5\{\\
&Positive:\ remove\ FileWriteData,\\
&Negative:\ remove\ relationship\ FileWriteData\ to\ k:\backslash\ \}
\end{aligned}$$

Although it is not possible to provide the user with a detailed set of instructions about how to perform the above, the information presented can be used to help make a better-informed decision on how to rectify the anomaly, and therefore go some way to help reduce the reliance on expert knowledge. The above mitigation information is informing the user that removing *FileWriteData* is identified as a positive effect to mitigation the identified anomaly. The effect of performing the action is that it will remove the relationship between $FileWriteData\ to\ k:\backslash$.

It should be noted that this technique is general in that it will provide suitable mitigation plans for anomalies identified in any *object-centric* dataset. This technique does have a degree of uncertainty as it is operating without domain-specific knowledge.

It is foreseen that the user will need to interpret the suggested mitigation action and determine its suitability. Although this is a limitation of the technique, it should be noted that providing a suggestion to a user with limited expert knowledge puts them in a better position to resolve security weaknesses than without using GraphBAD.

4 EXPERIMENTAL ANALYSIS

The aim of this analysis is to gain a comprehensive understanding of GraphBAD's ability to correctly identify both regular and irregular (anomalous) security data. Remarkably, there is a lack of security configuration benchmark datasets; however, anomaly detection audit logs are available. Therefore, we consider two different benchmark sets: (1) synthetically produced configuration datasets with varying degrees of complexity and number of anomalies, and (2) publicly available anomaly benchmark datasets used by other researchers. The latter allows us to make a direct comparison with other state-of-the-art techniques.

In the experimental analysis, performance is assessed using the following measures:

1. True Positive Rate (c): the fraction of irregular permissions correctly identified as irregular, and correctly included in the mitigation plan;
2. False Positive Rate (*fpr = 1 - tnr*): the fraction of regular permissions incorrectly identified as irregular, and incorrectly included in the mitigation plan;
3. True Negative Rate (*tnr*): the fraction of regular permissions correctly identified as regular, and correctly not included in the mitigation plan;
4. False Negative Rate (*fnr = 1- tpr*): the fraction of irregular permissions incorrectly classified as regular, and incorrectly not included in the mitigation plan;
5. *Accuracy* is reported as the fraction of all samples correctly identified. More specifically, $Accuracy = \frac{tpr+tnr}{tpr+tnr+fpr+fnr}$

It should be noted that all the measures consider, at the same time, the capability of GraphBAD in identifying anomalies and proposing an adequate mitigation plan.

GraphBAD has been programmed in 64-bit Java and executed on an Intel i7 with 16GB of RAM, equipped with Java 1.7. The software is multi-threaded to reduce processing time. For the examples presented in this paper, a maximum of 8 threads were used during the graph reduction and anomaly detection stage.

4.1 Synthetic datasets

In this section we firstly detail the algorithm developed for generating synthetic data, and then show the results of the experimental analysis performed on such data.

Algorithm 3 Algorithm for generated synthetic directory structures

Input: The maximum number of directories, $MaxDir$
Input: The step size of the directories, $StepDir$
Input: The maximum number of anomalies , $MaxAnom$
Input: The step size anomalies, $StepAnom$
Output: A set of directories, $S = \{s_1, s_2, ..., s_n\}$, where $s_n = \{d, p\}$ where d is the directory resource, and p is the permission level

```
1:  procedure GenerateData
2:      $belowT := \emptyset$
3:      $toCompute := V$
4:      repeat
5:          while $n \leftarrow 0$ do
6:              $i \leftarrow 0$
7:              while $i \leq MaxDir$ do
8:                  $S[j] \leftarrow createDirectory()$
9:                  $i+ = StepSize$
10:             end while
11:             $j \leftarrow 0, i \leftarrow 0$
12:             while $J \leq MaxAnom$ do
13:                 $dirNo \leftarrow genRandomInt(0, i)$
14:                 $pLevel \leftarrow genRandomInt(1, 14)$
15:                 $i+ = StepSize$
16:                 $S[j] \leftarrow pLevel$
17:             end while
18:             $n+ = StepAnom$
19:         end while
20:     until $n = MaxAnom$
21:     return $S$
22: end procedure
```

4.1.1 Generation of Synthetic Data

The synthetic datasets are generated by continuing with the example of auditing file system permissions. Different datasets have been created, and their characteristics are defined by the number of regular and irregular permissions. In our file system example, the dimensions are denoted by the number of directories, the number of users and the different levels of permissions. The number of irregular and regular permissions is programatically defined.

Algorithm 3 has been developed to generate synthetic datasets that closely replicate real-world security configurations [10]. The designed function takes four inputs: the maximum number of directories to create ($MaxDir$); the maximum number of anomalies to introduce ($MaxAnom$); the step size of the directories ($StepDir$); and the step size of the anomalies ($StepAnom$). The algorithm will produce a set of directory structures (S). Each set, s_i, will contain a sequence of directory resources and permissions, increasing in multiples of $StepDir$ until we reach the maximum directory size $MaxDir$. Line 5 is where the directory resource is created. However, for the purpose of creating synthetic datasets we do not need to create the actual directory structure, rather just a set representation. Each directory size will exist several times with a different frequency of simulated anomalies. The algorithm will start at $StepAnom$ and increase in multiples of $StepAnom$ until $MaxAnom$ is reached. The directory to assign the anomaly to, as well as the level of permission, are determined through generating a pseudo random number (lines 10 and 11). For the directory this will be between zero and the maximum number of directories ($MaxDir$), and the permission level will be established through generating a number between one and fourteen which is then translated into a permission level by setting the corresponding bits. For example, 1 denotes the permission level of full control, 2 to traverse folders, etc. A full definition of the individual attributes can be found in [9].

4.1.2 Empirical Results

For providing a comprehensive analysis of the system, we generate 380 unique datasets by using Algorithm 3 with the following four specifications:

1. $MaxDir$ = 1,000, $StepDir$ = 100, $MaxAnom$ = 9, $StepAnom$ = 1;
2. $MaxDir$ = 1,000, $StepDir$ = 100, $MaxAnom$ = 100, $StepAnom$ = 10;
3. $MaxDir$ = 10,000, $StepDir$ = 1000, $MaxAnom$ = 9, $StepAnom$ = 1; and
4. $MaxDir$ = 10,000, $StepDir$ = 1000, $MaxAnom$ = 100, $StepAnom$ = 10.

These specifications consider small to medium (100 to 1,000 directories) and large directory structure (1,000 to 11,000 directories). Test cases are created to simulate a small to medium (1 to 10) and a large number of anomalies (10 to 100). The motivation behind the creation of these datasets is to allow for a comprehensive analysis of GraphBAD against datasets with a different number of anomalies.

The Receiver Operating Characteristic space, illustrated in Figure 5, demonstrates the capability of GraphBAD to identify anomalies. Figure 5 illustrates the *tpr* and *fpr* for each dataset. The minimum accuracy ratio is 0.86; however, the average is above 0.98. The graph illustrates the trade-off between the *tpr* and *fpr*. The further the distance from the 45-degree (dotted) line, the more accurate the result. This clearly demonstrates GraphBAD's capability; especially considering that it has no prior knowledge regarding what is an anomaly, only the basic assumption regarding object relationships. It is worth emphasising the very low *fpr* value; given the typical scenario, where users are not required to be IT experts, it is very important that those identified as anomalies are actually anomalies, in order to minimise disruptions and changes. The average *fnr* and *tnr* are 0.58 and 0.98, respectively. The high *tnr* demonstrates the ability to correctly identify regular permissions as regular. The *fnr* has an average of 0.23, but it should be noted that the number of irregular permissions which are incorrectly classified as regular

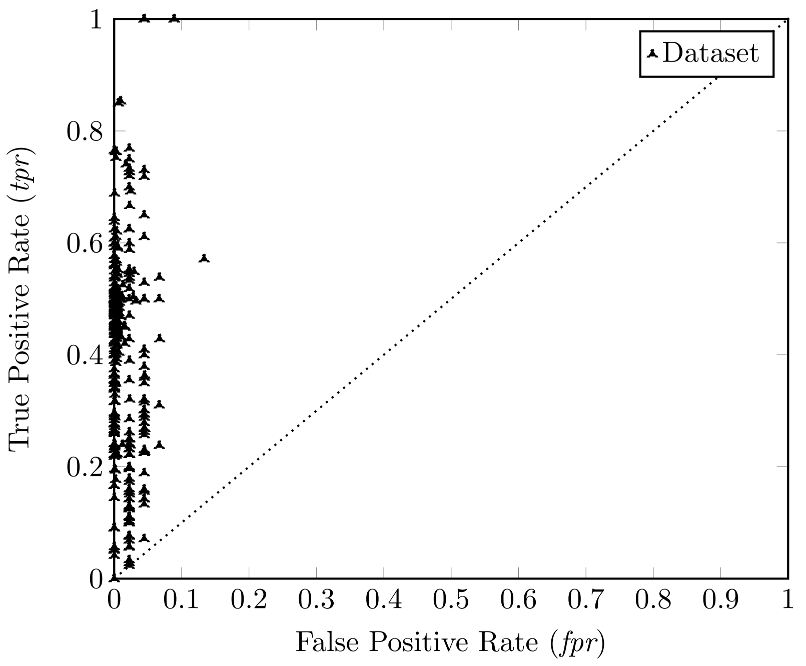

Figure 5: ROC space illustrating the *tpr* and *fpr* on the considered synthetic datasets.

increases as the number of anomalies also increase. This effect increases as the number of directories reduce. This is because the ratio of anomalies to regular permissions will decrease, making them less easy to identify by using the measure of regularity provided in this paper. However, it should be noted that considering GraphBAD is a very general approach, that does not exploit any specific application knowledge, these results are promising and show the potential of the technique.

Although Figure 5 demonstrates the classification characteristics of the technique, it is useful to also examine the relationship between the number of objects and the number of anomalies. Figure 6 (coloured) illustrates a polynomial surface (for result presentation only) of degree 5 for modelling the number of directories (objects), the number of anomalies (objects) and the accuracy. From this figure, it is noticeable that the accuracy is best when the number of anomalies is low and the number of directories is high. This is interesting as it demonstrates that the system is more effective at identifying anomalies when there is a large number of regular objects. However, this is no surprise as an increasing number of regular permissions will result in an increased measure of regularity, and therefore make the anomaly more easily identifiable. This clearly demonstrates that although the measure of regularity used in detecting subgraphs is simplistic, the results warrant its use. Figure 6 also illustrates an interesting trend where the accuracy decreases, and is at its lowest, when the number of anomalies is around 10, irrespective of directory size. However, this is potentially misleading as a large percentage of the generated datasets have less than 10 anomalies, and only very

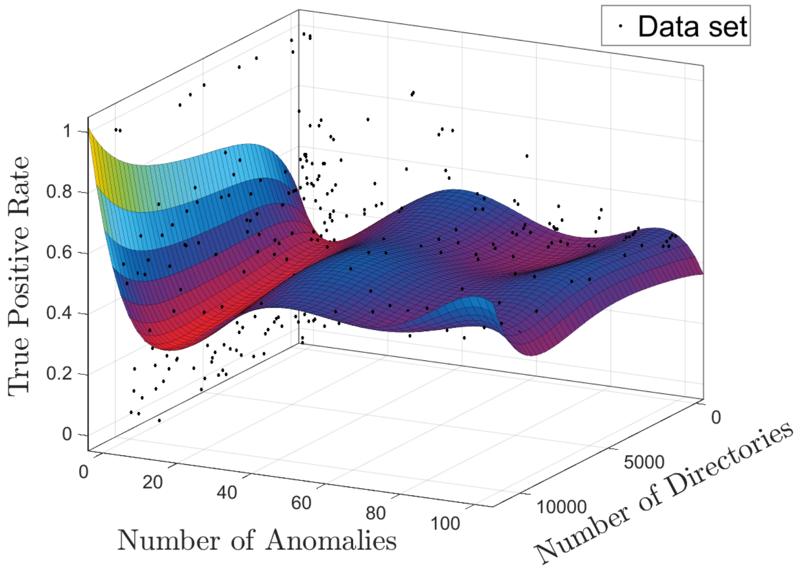

Figure 6: Polynomial fit illustrating how *accuracy* changes with respect to directory size and number of anomalies, on the considered synthetic data.

few have exactly 10.

In the application of examining file system permissions, GraphBAD is processing data offline and time is less of a concern. However, as GraphBAD ignores duplicate entries, it is able to maintain quick execution. The time required for execution is between 2-5 seconds for each of the synthetically generated datasets. Execution time increases as the number of anomalies also increases. The reason behind this increase is because the number of vertices and edges will increase with an increased variety of permissions (regular or anomalous).

4.2 Benchmarking

Even though research into anomaly detection is a large field, there is a lack of publicly available, standardised benchmarks for comparison of techniques [51]. The benchmark data that we are using is that from the Third International Knowledge Discovery and Data Mining Tools Competition (KDD Cup 1999 datasets) [8]. Although these data originates from 1999, many anomaly detection techniques have been evaluated using this data and, to the best of our knowledge, it is the most commonly used for bench marking. During this benchmarking, GraphBAD's accuracy is compared against other state-of-the-art techniques which have also been tested on the KDD datasets.

The KDD data are suitable for examining the presented technique as it comprises of a fixed set of connection-based features. The data is in Comma Separated Value (CSV) format, and each line contains a series of variables and an attack type. The structure of

Table 1: Survey of accuracy results from previous works using (part of) the KDD dataset.

Technique	**datasets derivation**	**Accuracy**
Unsupervised Cluster-based Detection [44]	datasets of 400k with 1-1.5% anomalies	0.48
γ-Algorithm [45, 46]	datasets of 2k with 5% anomalies	0.70
Subspace Categorical Data (Catsub) [47]	Entire KDD dataset divided into four attack categories	0.82
Real-Time Incremental Clustering (ADWICE) [48, 46]	Entire KDD dataset divided into four attack categories	0.95
Fuzzy C-Means (FCM)	Entire KDD dataset divided into four attack categories	0.96
Self-adaptive and Dynamic Clustering [49]	Divided into three attack types, refactored into three datasets with 99% normal and 1% anomaly	0.86
Genetic Algorithm & Support Vector Machine [50]	Random extraction of DOS attack only (no detail of size)	0.99

the data does not require any modification for use with GraphBAD. There are a total of 24 different attack types and “normal” in the KDD cup dataset. However, researchers who have previously used KDD bench marking data have raised concerns over the KDD Cup dataset which can bias comparative experiments [8]. Two identified limitations are: (1) the attack rate within the KDD Cup dataset is unnatural, and almost 80% of all instances correspond to an attack, and (2) the distribution of each attack type within the KDD cup dataset is unbalanced [45]. Another issue with using the KDD datasets for direct comparison is that other researchers often use different pre-processing techniques to generate sub datasets for evaluation. However, it should be noted that the KDD dataset, despite its limitations, has been widely used for benchmarking allows for state-of-the-art comparison. Table 1 details the results, and the considered KDD sub datasets, of state-of-the-art unsupervised anomaly detection algorithms tested on the KDD cup dataset.

As previously highlighted, the absence of any publicly available and standardised dataset makes it challenging to perform a systematic comparison with state-of-the-art in to unsupervised algorithms. One study determined that the use of a supervised algorithm could achieve 95% classification accuracy on KDD data [52], which is significantly better than the unsupervised techniques presented in Table 1. However, it should be noted that it is not realistic to compare unsupervised and supervised algorithms for the correct identification of irregular data items within the KDD dataset. Supervised algorithms have the benefit of being able to train to know *what* constitutes an irregular data entry through the use of classified training data, whereas unsupervised algorithms attempt to identify irregular data with no prior knowledge or training. Instead, GraphBAD is an

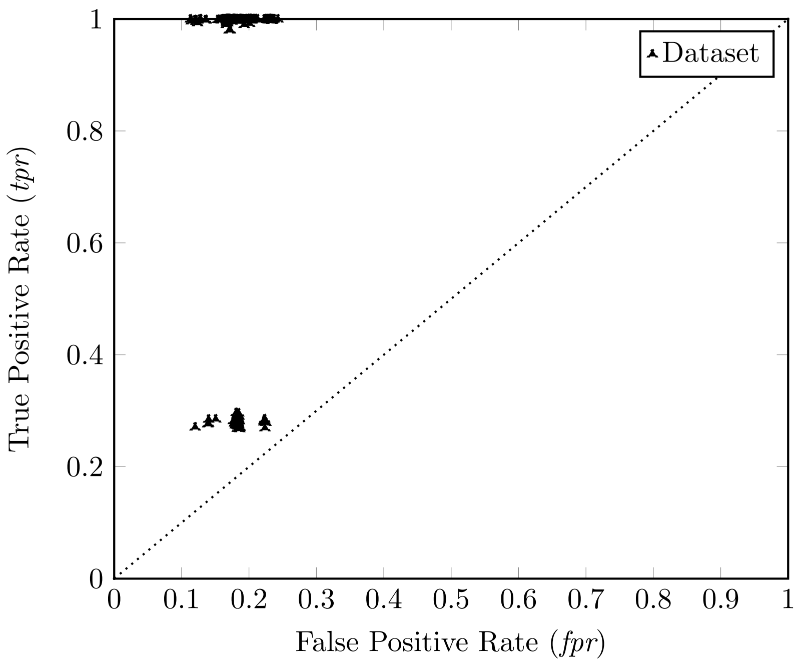

Figure 7: ROC space illustrating the *tpr* and *fpr* on the considered KDD datasets.

unsupervised algorithm for detecting anomalies in object-based datasets with no prior knowledge of *what* is classed as both normal and anomalous.

It should be noted here that a primary difference between GraphBAD and other approaches is that GraphBAD does not use any meta-data or restructuring techniques prior or during the processing of the KDD datasets. The use of meta-data would go against the aim of this work to produce a domain-independent technique for identifying security configuration anomalies. GraphBAD does not use any user input, only the actual data itself. It should also be noted that using the KDD data does not allow us to perform any mitigation planning. This is because the data does not represent a system's configuration, rather a network activity log.

In our experimental analysis, we test GraphBAD using the unlabeled KDD cup data which does not contain the attack type. Once processed, we consider the GraphBAD output against the labeled data –used as ground truth– in order to calculate *tpr*, *tnr* and *accuracy* as previously defined.

Although the techniques presented in Table 1 are unsupervised, each researcher has partitioned the data in different ways for processing. Specifically, the fact that only specific attack types have been selected potentially introduces bias. Here, in order to minimise potential bias and to provide a complete overview of performance, we decided to test GraphBAD on the complete KDD cup dataset (and a random sub-sample of 10% of it), and to systematically divide it into datasets with different characteristics. To provide a wide-scale analysis, the proposed methodology is to incrementally construct

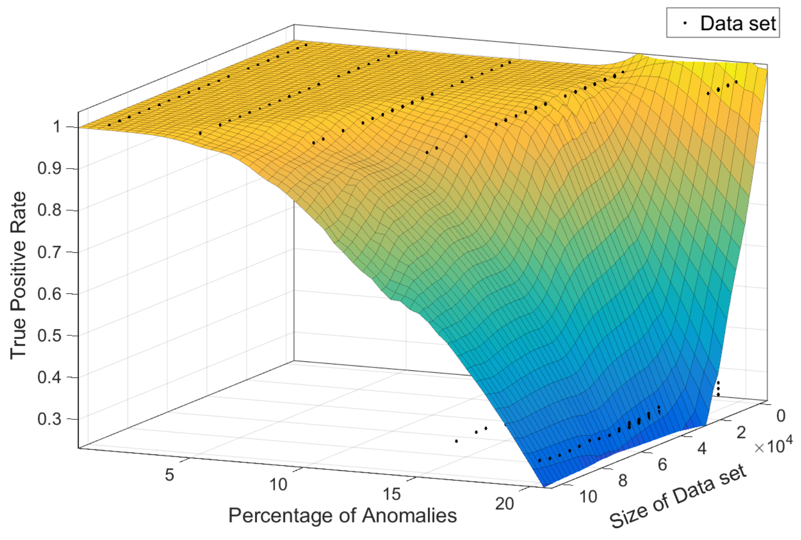

Figure 8: Polynomial fit illustrating how *accuracy* changes with respect to directory size and number of anomalies, on the considered KDD data.

datasets consisting of 1,000 entries, to 100,000 entries, in steps of 5,000. Each dataset will have an incrementally increasing number of attack entries from 1% to 20%, increasing by 5% in each increment. Both the normal and attack entries are extracted from the KDD cup dataset at random, and to get a complete representation, each data entry is only used one time.

In addition to the constructed subsets, we also process GraphBAD using both the 10% sample and the entire KDD cup dataset which results in an accuracy of 0.60 and 0.72, respectively. This performance is surprising as 80% of the data entries are attacks. However, the reason that GraphBAD still has such a high accuracy rate is because the 80% attack data is further divided into 24 different attack types, making an average of less than 4% for each attack. The results demonstrate that 20% of the data being normal is enough for GraphBAD to achieve a good accuracy rate.

The execution time of GraphBAD deteriorates as the size of the data structure increases. For example, execution time for datasets containing 1,000 and 100,000 entries were 5 and 6834 seconds (1.8 hours), respectively. This demonstrates that GraphBAD would suffer from poor performance on datasets of a large size; however, a dataset with over 100,000 datasets would represent a large security configuration. Furthermore, the time required for GraphBAD to analyse a system configuration is insignificant compared to the time required for a human expert to thoroughly analyse the configuration.

The results presented in Table 2 demonstrate the average results of GraphBAD on the KDD datasets generated using the previously described methodology. The results in the table centre around the percentage of anomalies (attacks), rather than the size of the dataset. We chose to present the results in this way as the results only differ slightly

Table 2: GraphBAD performance, on the KDD sub-sets, with respect to different anomaly percentages. Results (averaged) are shown in terms of accuracy, true positive rate, and false positive rate.

Anomaly Percentage	**tpr**	**fpr**	**tnr**	**fnr**	**Accuracy**
1%	1.00	0.17	0.00	0.83	0.84
5%	1.00	0.17	0.00	0.83	0.83
10%	1.00	0.20	0.00	0.80	0.82
15%	0.95	0.23	0.05	0.77	0.80
20%	0.45	0.45	0.55	0.55	0.74

depending upon dataset size. For example, the accuracy for 1% of anomalies in dataset sizes of 1000 and 20,000 is 0.87 and 0.81, respectively. It is also noticeable from the results that the *tpr* is at 100% until the percentage of attacks reaches 15%. The *fpr* is at 17% with both 1% and 5% of attacks and increases to 45% with 20% of attacks. The *fpr* is the main weakness of GraphBAD as it describes the percentage of regular data entries which are identified as irregular, i.e., the system is too sensitive. Figure 7 illustrates the ROC space where it is noticeable there are two distinct clusters: those with a high *trp*, and those with a low *tpr*. Figure 8 (coloured) illustrates a surface fit with a polynomial surface of degree 5. From this it is possible to deduce that those with a low *tpr* are those with a high number of data entries ($> 15,000$) with a high percentage of anomalies ($> 10\%$). The results in the table demonstrate that the accuracy of the proposed system diminishes with an increasing percentage of anomalies. To the best of our knowledge, results shown in Table 2 and those achieved on the complete KDD dataset indicate that GraphBAD can provide better results than existing unsupervised approaches. In fact, the only two other approaches (first two rows of Table 1) that consider all the attack types –though only on a subset of the KDD data– show significantly worse performance.

5 Conclusion

Auditing security configurations is a complex task, that heavily relies on expert knowledge, which is required for understanding the different security configurations. However, expert knowledge can be incomplete or erroneous, and the cost of security experts can be unaffordable for home users and small companies.

For addressing the above issues, in this paper we presented the novel Graph-based Security Anomaly Detection (GraphBAD) approach. GraphBAD does not require any previous knowledge on the data structure; the only assumptions are that security data are provided in text form, and that objects in the same dataset are related. GraphBAD converts security configuration and audit log data into a graph-based model. The graph-based model is then analysed in order to identify anomalous (irregular) subgraphs, which possibly indicate security issues. A mitigation plan is then provided; it allows to disable an anomaly, aiding the user in rectifying the problem with minimal expert knowledge.

Beside the design and development of GraphBAD, and the definition of anomalies in the given context, the main contributions of this paper are: (i) a new method for generating synthetic datasets, that will be made publicly available in order to increase the currently limited number of benchmarks available for researchers; (ii) a large experimental analysis, performed on both synthetically generated data, and the well-known KDD dataset. Specifically, the experimental analysis demonstrated the good (average of 98% for synthetic, 80% for KDD) accuracy performance of GraphBAD. The experimental analysis presented in this paper demonstrates the suitability of GraphBAD to identify anomalies in large data structures where up to 20% of their content is anomalous. A system with such a high number of configuration errors would be classes as having a substantial number of vulnerabilities. However, it should be noted that the accuracy of the system deteriorates as the number of anomalies increases. It can be concluded that GraphBAD performs best of datasets that contain a lower number of anomalies.

The purpose of GraphBAD is to provide a general technique to identify irregularities in object-based configuration data. In pursuing this ambition, a measure of regularity has been adapted that has been empirically evaluated on both KDD dataset and synthetic file system permissions, which are representative of other object-based data sources. A limitation is that this measure could be ineffective for processing data sources that have different data characteristics. It is likely that the measure will need to be changed to improve performance in some instances; however, this is likely to be influenced by domain knowledge and will go against the criteria of GraphBAD being domain independent.

The accuracy characteristics of GraphBAD acquired from performing empirical evaluation on both synthetic file system configuration and KDD datasets has demonstrated suitability in the technical implementation. However, there is a possibility that GraphBAD may suffer adverse accuracy characteristics when processing datasets containing anomalies of different types. In this instance, the user may need to adjust the threshold value to improve accuracy. It is also possible that anomalies in datasets that result in a sparse graph would be undetected as there is no distinguishable numeric difference in measure of regularity between both regular and irregular datasets. However, despite these potential limitations, GraphBAD is unsupervised and the empirical analysis performing in this paper has demonstrates its good applicability to different datasets.

We see several avenues for future work. Firstly, we plan to design a technique capable of automatically identifying object relationships within both structured and unstructured data. Secondly, a further empirical evaluation –involving real-world data– is devised. Thirdly, we are interested in testing the proposed approach in different domains that allows to analyse structured data by considering an undirected graph encoding. This may lead to the development of different measures of regularity which give improved performance and accuracy on different datasets. Finally, we would like to emphasise that the proposed approach can also be used either as a training or support tool for security experts. For this reason, we aim to develop a graphical user interfacein order to make GraphBAD more accessible and more easily exploitable. We also foresee the need to undertake further research to provide technological support tools which can help users interpret and utilise the provided mitigation plan.

6 Availability

All source code, datasets and results presented in this paper are available at:
`https://selene.hud.ac.uk/scomsp2/GraphBad.zip`

References

[1] J. Hoffmann, "Simulated penetration testing: From "Dijkstra" to "Turing Test++"," in *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[2] J. L. Obes, C. Sarraute, and G. Richarte, "Attack planning in the real world," in *Working Notes for the 2010 AAAI Workshop on Intelligent Security (SecArt)*, p. 10, 2010.

[3] S. Sohrabi, O. Udrea, and A. Riabov, "Hypothesis exploration for malware detection using planning," in *Proceedings of the 27th National Conference on Artificial Intelligence (AAAI)*, pp. 883–889, 2013.

[4] A. V. Riabov, S. Sohrabi, D. Sow, D. Turaga, O. Udrea, and L. Vu, "Planning-based reasoning for automated large-scale data analysis," in *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[5] HM Government, "Requirements for basic technical protection from cyber attacks," 2014.

[6] U. Franke and J. Brynielsson, "Cyber situational awareness–a systematic review of the literature," *Computers & Security*, vol. 46, pp. 18–31, 2014.

[7] S. L. Garfinkel, "Digital forensics research: The next 10 years," *digital investigation*, vol. 7, pp. S64–S73, 2010.

[8] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, 2009.

[9] S. Parkinson, V. Somaraki, and R. Ward, "Auditing file system permissions using association rule mining," *Expert Systems with Applications*, vol. 55, pp. 274–283, 2016.

[10] S. Parkinson and A. Crampton, "Identification of irregularities and allocation suggestion of relative file system permissions," *Journal of Information Security and Applications*, 2016.

[11] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and big heterogeneous data: a survey," *Journal of Big Data*, vol. 2, no. 1, pp. 1–41, 2015.

[12] R. Luh, S. Marschalek, M. Kaiser, H. Janicke, and S. Schrittwieser, "Semantics-aware detection of targeted attacks: a survey," *Journal of Computer Virology and Hacking Techniques*, pp. 1–39, 2016.

[13] R. Gabriel, T. Hoppe, A. Pastwa, and S. Sowa, "Analyzing malware log data to support security information and event management: Some research results," in *Advances in Databases, Knowledge, and Data Applications, 2009. DBKDA '09. First International Conference on*, pp. 108–113, March 2009.

[14] P. Dairinram, D. Wongsawang, and P. Pengsart, "Siem with lsa technique for threat identification," in *2013 19th IEEE International Conference on Networks (ICON)*, pp. 1–6, Dec 2013.

[15] S. Asanger and A. Hutchison, "Experiences and challenges in enhancing security information and event management capability using unsupervised anomaly detection," in *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pp. 654–661, Sept 2013.

[16] V. Shandilya, C. B. Simmons, and S. Shiva, "Use of attack graphs in security systems," *Journal of Computer Networks and Communications*, vol. 2014, 2014.

[17] A. A. E. Elhadi, M. A. Maarof, and A. H. Osman, "Malware detection based on hybrid signature behaviour application programming interface call graph," *American Journal of Applied Sciences*, vol. 9, no. 3, p. 283, 2012.

[18] B. J. Kwon, J. Mondal, J. Jang, L. Bilge, and T. Dumitras, "The dropper effect: Insights into malware distribution with downloader graph analytics," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1118–1129, ACM, 2015.

[19] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, (New York, NY, USA), pp. 217–224, ACM, 2002.

[20] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *Proceedings of the 1998 workshop on New security paradigms*, pp. 71–79, ACM, 1998.

[21] O. Sheyner and J. Wing, "Tools for generating and analyzing attack graphs," in *Formal methods for components and objects*, pp. 344–371, Springer, 2003.

[22] E. A. Manzoor, S. Momeni, V. N. Venkatakrishnan, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," *arXiv preprint arXiv:1602.04844*, 2016.

[23] M. S. Boddy, J. Gohde, T. Haigh, and S. A. Harp, “Course of action generation for cyber security using classical planning.,” in *ICAPS*, pp. 12–21, 2005.

[24] J. L. Obes, C. Sarraute, and G. Richarte, “Attack planning in the real world,” *arXiv preprint arXiv:1306.4044*, 2013.

[25] C. C. Noble and D. J. Cook, “Graph-based anomaly detection,” in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’03, (New York, NY, USA), pp. 631–636, ACM, 2003.

[26] W. Eberle and L. Holder, “Discovering structural anomalies in graph-based data,” in *Data Mining Workshops, 2007. ICDM Workshops 2007. Seventh IEEE International Conference on*, pp. 393–398, Oct 2007.

[27] A. Inokuchi, T. Washio, and H. Motoda, “An apriori-based algorithm for mining frequent substructures from graph data,” in *Principles of Data Mining and Knowledge Discovery*, pp. 13–23, Springer, 2000.

[28] X. Yan and J. Han, “gspan: Graph-based substructure pattern mining,” in *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pp. 721–724, IEEE, 2002.

[29] P. Bangcharoensap, H. Kobayashi, N. Shimizu, S. Yamauchi, and T. Murata, *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part III*, ch. Two Step graph-based semi-supervised Learning for Online Auction Fraud Detection, pp. 165–179. Cham: Springer International Publishing, 2015.

[30] S. Parkinson and D. Hardcastle, “Automated planning for file system interaction,” in *32nd Workshop of the UK Planning and Scheduling Special Interest Group*, December 2014.

[31] 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.

[32] M. Roberts, A. Howe, I. Ray, M. Urbanska, Z. S. Byrne, and J. M. Weidert, “Personalized vulnerability analysis through automated planning,” in *Working Notes of IJCAI 2011, Workshop Security and Artificial Intelligence (SecArt-11)*, vol. 4, 2011.

[33] V. Bandara, A. Pezeshki, and P. J. Anura, “Modeling spatial and temporal behavior of internet traffic anomalies,” in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pp. 384–391, IEEE, 2010.

[34] S. Ali, *Enabling Accurate Anomaly Detection Using Progressive Security-Aware Packet Sampling.* PhD thesis, School of Electrical Engineering and Computer Science, National University of Sciences and Technology, 2009.

[35] W. Budgaga, M. Malensek, S. Lee Pallickara, and S. Pallickara, “A framework for scalable real-time anomaly detection over voluminous, geospatial data streams,” *Concurrency and Computation: Practice and Experience*, vol. 29, no. 12, 2017.

[36] G. Gu, J. Zhang, and W. Lee, “Botsniffer: Detecting botnet command and control channels in network traffic.,” in *NDSS*, vol. 8, pp. 1–18, 2008.

[37] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee, “Bothunter: Detecting malware infection through ids-driven dialog correlation.,” in *USENIX Security Symposium*, vol. 7, pp. 1–16, 2007.

[38] J. G. Alfaro, N. Boulahia-Cuppens, and F. Cuppens, “Complete analysis of configuration rules to guarantee reliable network security policies,” *International Journal of Information Security*, vol. 7, no. 2, pp. 103–122, 2008.

[39] M. Gander, M. Felderer, B. Katt, A. Tolbaru, R. Breu, and A. Moschitti, “Anomaly detection in the cloud: Detecting security incidents via machine learning.,” in *EternalS@ ECAI*, pp. 103–116, Springer, 2012.

[40] S. Parkinson, V. Somaraki, and R. Ward, “Auditing file system permissions using association rule mining,” *Expert Systems with Applications*, vol. 55, pp. 274–283, 2016.

[41] A. Sapegin, M. Gawron, D. Jaeger, F. Cheng, and C. Meinel, “Evaluation of in-memory storage engine for machine learning analysis of security events,” *Concurrency and Computation: Practice and Experience*, vol. 29, no. 2, 2017.

[42] S. Parkinson and A. Crampton, “A novel software tool for analysing NT file system (NTFS) permissions,” *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 6, pp. 266–272, 2013.

[43] J. Liu, E. Bier, A. Wilson, T. Honda, S. Kumar, L. Gilpin, J. Guerra-Gomez, and D. Davies, “Graph analysis for detecting fraud, waste, and abuse in healthcare data,” in *Twenty-Seventh IAAI Conference*, 2015.

[44] L. Portnoy, E. Eskin, and S. Stolfo, “Intrusion detection with unlabeled data using clustering,” in *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*, Citeseer, 2001.

[45] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, “Learning intrusion detection: Supervised or unsupervised?,” in *Image Analysis and Processing – ICIAP 2005* (F. Roli and S. Vitulano, eds.), vol. 3617 of *Lecture Notes in Computer Science*, pp. 50–57, Springer Berlin Heidelberg, 2005.

[46] P. Gogoi, B. Borah, and D. K. Bhattacharyya, “Anomaly detection analysis of intrusion data using supervised & unsupervised approach,” *Journal of Convergence Information Technology*, vol. 5, no. 1, pp. 95–110, 2010.

[47] B. Borah and D. Bhattacharyya, “Catsub: a technique for clustering categorical data based on subspace,” *J Comput Sci*, vol. 2, pp. 7–20, 2008.

[48] K. Burbeck and S. Nadjm-Tehrani, “Adwice–anomaly detection with real-time incremental clustering,” in *Information Security and Cryptology–ICISC 2004*, pp. 407–424, Springer, 2005.

[49] S. Lee, G. Kim, and S. Kim, “Self-adaptive and dynamic clustering for online anomaly detection,” *Expert Systems with Applications*, vol. 38, no. 12, pp. 14891 – 14898, 2011.

[50] D. S. Kim, H.-N. Nguyen, and J. S. Park, “Genetic algorithm to improve SVM based network intrusion detection system,” in *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, vol. 2, pp. 155–158 vol.2, AINA, March 2005.

[51] A. F. Emmott, S. Das, T. Dietterich, A. Fern, and W.-K. Wong, “Systematic construction of anomaly detection benchmarks from real data,” in *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, pp. 16–21, ACM, 2013.

[52] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, “Learning intrusion detection: supervised or unsupervised?,” *Image Analysis and Processing–ICIAP 2005*, pp. 50–57, 2005.