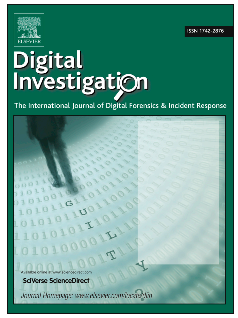
https://doi.org/10.1016/j.diin.2017.05.001

Accepted Manuscript



Graph clustering and anomaly detection of access control log for forensic purposes

Hudan Studiawan, Christian Payne, Ferdous Sohel

PII: S1742-2876(17)30143-3

DOI: 10.1016/j.diin.2017.05.001

Reference: DIIN 679

To appear in: *Digital Investigation*

Received Date: 15 July 2016

Revised Date: 21 March 2017

Accepted Date: 1 May 2017

Please cite this article as: Studiawan H, Payne C, Sohel F, Graph clustering and anomaly detection of access control log for forensic purposes, *Digital Investigation* (2017), doi: 10.1016/j.diin.2017.05.001.

Graph Clustering and Anomaly Detection of Access Control Log for Forensic Purposes

Hudan Studiawan[a,b,*], Christian Payne[a], Ferdous Sohel[a]

[a]*School of Engineering and Information Technology, Murdoch University, Australia*
[b]*Department of Informatics, Institut Teknologi Sepuluh Nopember, Indonesia*

Abstract

Attacks on operating system access control have become a significant and increasingly common problem. This type of security threat is recorded in a forensic artifact such as an authentication log. Forensic investigators will generally examine the log to analyze such incidents. An anomaly is highly correlated to an attacker's attempts to compromise the system. In this paper, we propose a novel method to automatically detect an anomaly in the access control log of an operating system. The logs will be first preprocessed and then clustered using an improved MajorClust algorithm to get a better cluster. This technique provides parameter-free clustering so that it automatically can produce an analysis report for the forensic investigators. The clustering results will be checked for anomalies based on a score that considers some factors such as the total members in a cluster, the frequency of the events in the log file, and the inter-arrival time of a specific activity. We also provide a graph-based visualization of logs to assist the investigators with easy analysis. Experimental results compiled on an open dataset of a Linux authentication log show that the proposed method achieved the accuracy of 83.14% in the authentication log dataset.

Keywords: authentication log, improved MajorClust, event log forensics, anomaly detection

*Corresponding author
Email addresses: hudan@if.its.ac.id (Hudan Studiawan), c.payne@murdoch.edu.au (Christian Payne), f.sohel@murdoch.edu.au (Ferdous Sohel)

1. Introduction

Access controls limit what actions user can perform in a specific environment such as an application or operating system. In this research, we mainly consider the access controls in an operating system. For example, this log is available in `auth.log` under the `/var/log/` directory in a Debian-based Linux environment or `/var/log/secure` in RedHat family distributions. It provides a log of Pluggable Authentication Modules (PAM) that record the user's granted access [1] and also secure shell (SSH) accesses, both failed and successful ones. This artifact is important evidence for the forensic investigators or security analysts to trace the attackers and analyze the incidents in a server.

For `auth.log` analysis, there is `OSSEC` (Open Source Host-based Intrusion Detection System Security), which also considers other log files to detect suspicious activities in a host [2]. Some examples of OSSEC implementation to detect access violations and multiple failed logins were presented in [3] while its complete guide is described in [4]. Other research on `auth.log` investigation was proposed by Sato and Yamauchi [5]. They first provided an architecture for securing log files and then proposed tampering detection of system logs including `auth.log`. In addition, Basin et al. gave a brief explanation of possible attacks and a simple examination using existing tools to check the traces for attempted attacks. As a result, we can see a trail in the security log such as `auth.log` [6].

One of the applications that record activities in `auth.log` is the SSH server. Detection of brute force attacks is critical because recent reports show this type of attack is frequently conducted where SSH service is widely targeted [7]. Instead of doing forensic analysis, most research have conducted a proactive mechanism to prevent the dictionary attack [8, 9]. The authors presented a detection and defense architecture for SSH threats based on the authentication log. Another attempt to create a model for brute force profiling was proposed by Javed and Paxson where the model accounted the detection and user authentication failures [10].

The application of text clustering in digital evidence has also been proposed to increase the relevance of the search results [11]. Several papers have tried to cluster the event log and detect its outliers, but the threshold for the anomaly scores was not sophisticated since the user must tune it manually to get the best results [12, 13].

In this paper, we consider SSH records as well as PAM authentication in `auth.log`. The proposed method identifies any log activities that are suspicious and labels them as outliers after the clustering phase. We propose a parameter-free algorithm to cluster this log and conduct anomaly detection, which refers to establish a baseline norm and detecting deviations from it. These deviations can be an attack or other strange events which need attention from the forensic analysts. However, the experiments focus on brute force password attacks as this is the most common type of access control violation [7]. The term *parameter-free* means that the clustering algorithm will run without any initial guess or any parameter supplied.

Most existing techniques need a manual input of the number of clusters. The classical k-Means and its variants still need the number of clusters as a parameter to run [14] while hierarchical clustering requires a specific level at which to cut the generated tree or dendrogram [15]. Density-based clustering and its extensions are other alternatives, but they require the neighborhood size to be passed as a user-defined variable [16].

Another technique, namely MajorClust, is a parameterless method that clusters nodes based on its natural structure in the graph [17]. MajorClust has been extended to a fuzzy version [18] and probability-based approach, so it can work well with several structures on the local scale [19]. However, we will improve MajorClust by adding a condition when the processed node's cluster is not the same as the heaviest neighbor node's cluster, we will force it to follow the cluster of the heaviest one. Therefore, an event will stick to the most similar one instead of following the others, which are not so similar but has more edge weight aggregation. This will improve the performance of clustering so that it will be suitable with the log record. The next section will describe the proposed method based on refined MajorClust.

2. Proposed method

We define some formal terms in this paper. An *event log* or a *log file* is a file that records activities from an application or services such as authentication log from SSH server application. A *record* is defined as a single entry in the event log. Then, an *event* is defined as a message in a record.

The proposed method is illustrated in Figure 1. First, the raw log will be preprocessed and converted into a graph model. Second, we enter the clustering phase which consists of four steps:

1. Cluster graph using improved MajorClust (Phase 1);
2. Create new representation of graph based on multiple longest common substring (MLCS) to solve the overfitting problem (Phase 2);
3. Refine the new form of cluster by running improved MajorClust once again (Phase 3);
4. Produce clustering result using initial graph representation (Phase 4).

After that, we conduct anomaly calculation on the clustering results based on a score considering several properties that characterize each cluster. An estimated threshold for anomalousness is provided to detect this behavior automatically. The last step is to provide a visualization of the detected outliers. The details of each phase are described in the following subsections.

2.1. Log preprocessing and proposed graph model

In this paper, the digital evidence to be investigated is the authentication log file (`auth.log`) in a Linux environment. It is assumed that the log file has not been tampered with or modified by the attackers. The full set of features for analysis are datetime, hostname, service or process name, process identifier (PID), and the event. The feature used in log preprocessing step is only the event since we focus on the content of the log and we do not consider other fields. It should be noted that we do not delete any fields from the log, rather

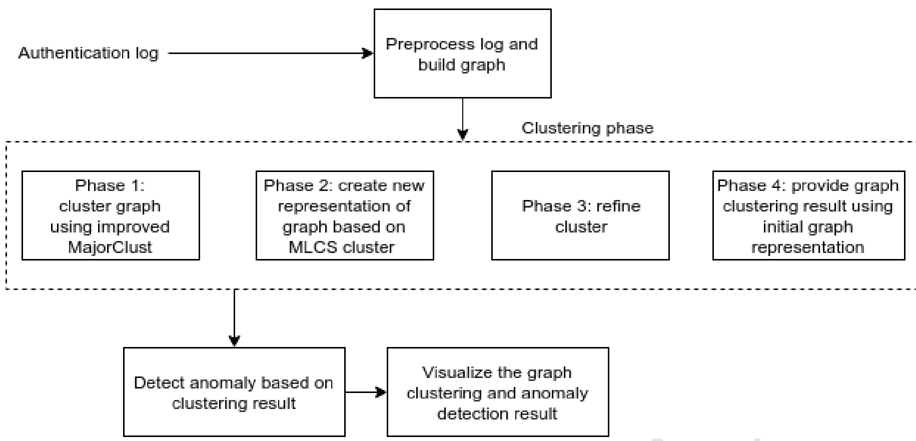


Figure 1: Block diagram of proposed method

only temporarily omit these features in the clustering process. For example, we need datetime property for calculating the anomaly score later. Other ignored properties include the hostname of the server, type of log and its process ID (PID), e.g., sshd[2790] and CRON[2839]. We filter all events, so it will produce only the unique one and attach the row log identifier to every unique event.

Furthermore, all numeric characters and stopwords in unique events are removed. We include some additional words that are not included in standard English stopwords but exist in the log record, e.g., *preauth, from, for, port, sshd, ssh,* and *root*. To identify non-standard stopwords, we conducted the experiments and checked the results of the graph clustering. If the results are not formed well, we analyze the string message and add the stopwords in order to increase the clustering accuracy. The added stopwords commonly appear in the string and if not removed it will send the different messages into one cluster.

Moreover, a $tf{-}idf$ (term frequency−inverse document frequency) procedure is implemented in every filtered line to produce its numerical representation. Note that in our case, document d refers to a single line of log record l. First, term frequency, tf, is not only the number of occurrences of term t in log l but it is normalized with the total number of terms in l, which is denoted as

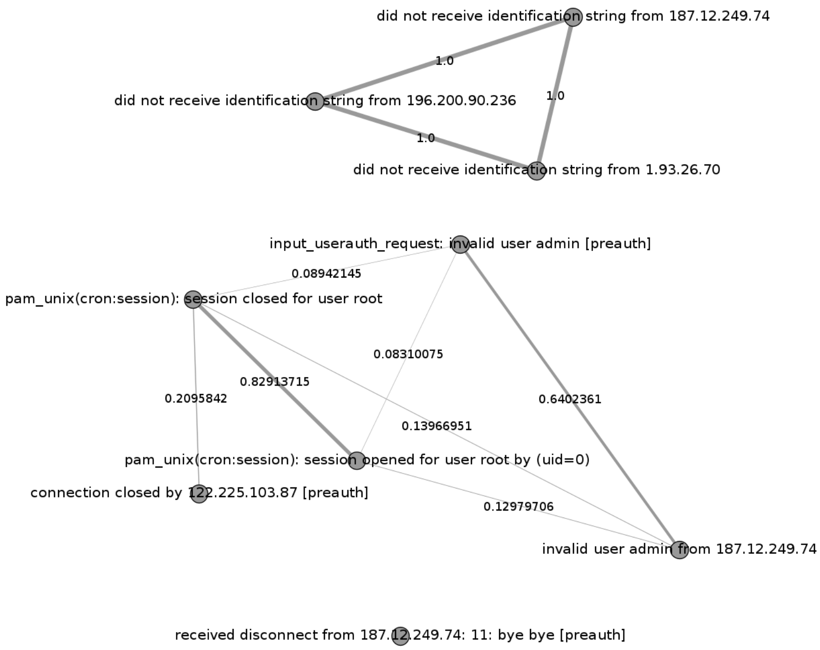


Figure 2: Proposed graph model for authentication log

$len(l)$ so that:

$$tf_{t,l} = \frac{tf}{len(l)} \tag{1}$$

Second, we need to calculate *inverse document frequency* of term t, idf_t, as shown in the equation below:

$$idf_t = 1 + \log \frac{N}{df_t} \tag{2}$$

where N is the number of lines in the event log, and df_t is the total number of lines in which the term occurs. We modify a basic formula of $tf_{t,l}$ and idf_t from [20] to conform with the real event log and avoid a zero value. Finally, we calculate $tf{-}idf$ of term t as follows:

$$tf{-}idf_{t,l} = tf_{t,l} \times idf_t \tag{3}$$

The next step is to build a graph $G = (V, E, w)$ where V, a set of vertices, represents each unique event, E, a set of edges, depicts relationship between two vertices where its weight, w, is the cosine distance to measure the similarity between two events in a single log record l as seen below [20]:

$$w(l_1, l_2) = \frac{\vec{V}(l_1) \cdot \vec{V}(l_2)}{|\vec{V}(l_1)||\vec{V}(l_2)|} \tag{4}$$

where the numerator denotes the dot product of the vector $\vec{V}(l_1)$ and $\vec{V}(l_2)$ which is represented as $\sum_{i=0}^{p} \vec{V_i}(l_1)\vec{V_i}(l_2)$. On the other hand, the denominator is the product of their Euclidean lengths and defined as $\sqrt{\sum_{i=0}^{d} \vec{V_i}^2(l)}$ where d is the total number of the term calculated in a record l. This cosine similarity distance becomes the edge weight of two vertices. An example of the proposed graph model is shown in Figure 2. We can see that every vertex is connected to other vertices except its zero distance since the edge is only created when the cosine similarity is greater than zero.

To provide a more clear illustration of cosine similarity in the event log, we give a step by step example of how this measurement is calculated between two events as follow.

Step 1: Two events from an event log

Dec 1 23:05:20 ip-172-31-27-153 sshd[28547]: Invalid user test from 192.208.179.82

Dec 1 23:36:42 ip-172-31-27-153 sshd[28578]: Invalid user admin from 187.76.79.142

Step 2: Preprocessing

l_1: invalid user test

l_2: invalid user admin

Step 3: Calculating term-frequency (tf)

Event	invalid	user	test	admin
l_1	1	1	1	0
l_2	1	1	0	1

Step 4: Calculating normalized term-frequency (tf) based on Equation 1

Event	invalid	user	test	admin
l_1	0.3	0.3	0.3	0
l_2	0.3	0.3	0	0.3

Step 5: Calculating inverse document frequency (idf) based on Equation 2

$idf_{invalid} = 1 + \log 2/2 = 1$

$idf_{user} = 1 + \log 2/2 = 1$

$idf_{test} = 1 + \log 2/1 = 0.693$

$idf_{admin} = 1 + \log 2/1 = 0.693$

Step 6: Calculating $tf - idf$ based on Equation 3

Event	invalid	user	test	admin
l_1	0.3*1	0.3*1	0.3*0.693	0
l_2	0.3*1	0.3*1	0	0.3*0.693

Event	invalid	user	test	admin
l_1	0.3	0.3	0.208	0
l_2	0.3	0.3	0	0.208

Step 7: Calculating the numerator of cosine similarity

$|\vec{V}(l_1)| \cdot |\vec{V}(l_2)|$ = 0.3*0.3 + 0.3*0.3 + 0.208*0 + 0*0.208

= 0.09 + 0.09 + 0 + 0

= 0.18

Step 8: Calculating the denominator of cosine similarity

$|\vec{V}(l_1)| = sqrt(0.3^2 + 0.3^2 + 0.208^2 + 0^2)$

$= sqrt(0.09 + 0.09 + 0.043 + 0)$

$= sqrt(0.223)$

$= 0.472$

$|\vec{V}(l_2)| = sqrt(0.3^2 + 0.3^2 + 0^2 + 0.208^2)$

$= sqrt(0.09 + 0.09 + 0 + 0.043)$

$= sqrt(0.223)$

$= 0.472$

Step 9: Calculating the cosine similarity based on Step 7, 8, and Equation 4

$w(l_1, l_2)$ = numerator / denominator

= 0.18 / (0.472 * 0.472)

= 0.18 / 0.228

= 0.789

2.2. Event log clustering based on improved MajorClust

The first phase to detect the anomaly is to cluster the generated graph using the MajorClust algorithm. For the sake of completeness, we include a brief technical description of MajorClust. For details of the algorithm, the reader is referred to [17]. Initially, each node is attached to its own cluster. For each vertex, the procedure will accumulate the edge weight of the neighboring nodes as denoted below:

$$c_i = \sum_{j=1}^{n} w(e_{i_j}),\ 0 \le i \le m \quad (5)$$

where c_i is the i^{th} neighboring clusters, $w(e)$ is the edge weight for each neighboring node in a particular cluster, m is the total of the neighboring cluster i, and n is the total member of vertices in the cluster c_i. The next step is assigning each vertex to a cluster, which has maximum weight aggregation and is defined as:

$$c^* = \arg\max_i(c_i) \quad (6)$$

This process is continued until all possible combination checking for current and assigned clusters for each node is accomplished.

However, we identified a major drawback in the original MajorClust technique when we applied it in a `auth.log` file. It will fail to cluster a large number

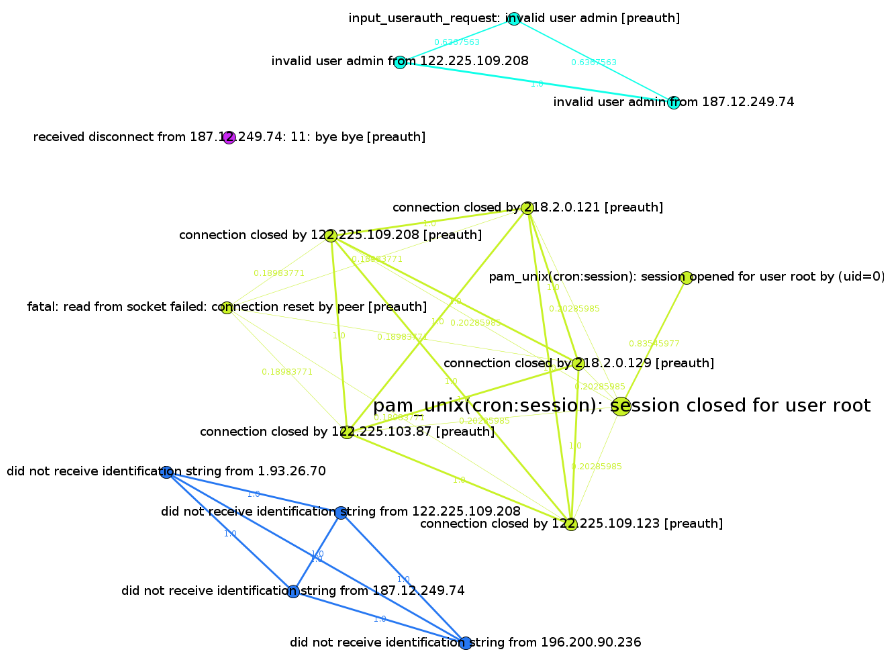


Figure 3: The drawback of MajorClust with more vertices

of nodes since it only depends on the accumulated weight of the neighboring vertices. A node will only depend on the aggregate weight of the neighbors although it is less relevant and does not consider the more similar cosine distance.

This shortcoming is illustrated in Figure 3 and indicated by a large green node with label `pam_unix(cron:session): session closed for user root`. This node should create its own cluster since it is closer with another vertex with a more similar event.

Therefore, we improve the MajorClust by supplying the additional requirement that when the current cluster is not the same as the heaviest neighbor node, we force the operating node to follow the cluster of the heaviest one. The heavier the edge weight, the closer the distance between two events in the log record since it represents the cosine similarity. By deploying this improvement, an event will stick to the most similar one instead of following the others, which are not so similar but has more edge weight aggregation. We then refine

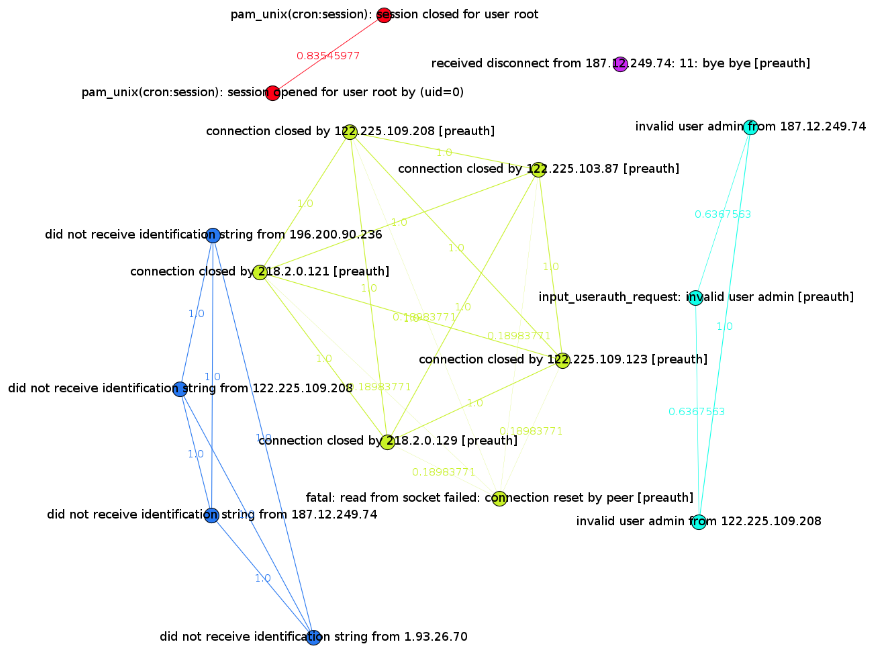


Figure 4: The improvement of MajorClust algorithm

Equation (6) as follows:

$$\mathbf{c} = \begin{cases} \arg\max_i(c_i) & \text{if } c^* = c_i(h) \\ c_i(h) & \text{if } c^* \neq c_i(h) \end{cases} \tag{7}$$

where $\mathbf{c}$ is the current cluster considering additional checking, h is a neighbor node with the heaviest edge weight, and $c(h)$ is the cluster label of h. The illustration of this improvement is depicted in Figure 4. We can see that the node in the previous figure has created a new cluster as expected (indicated by vertices in red).

Nevertheless, this improvement triggers another problem. The generated cluster is overfitting since it produces many small clusters with only two or three vertices as shown in Figure 5. This issue was found when the proposed procedure processed the first 500 records of the first-day log in the dataset. We solve this problem by running one additional round of MajorClust called the

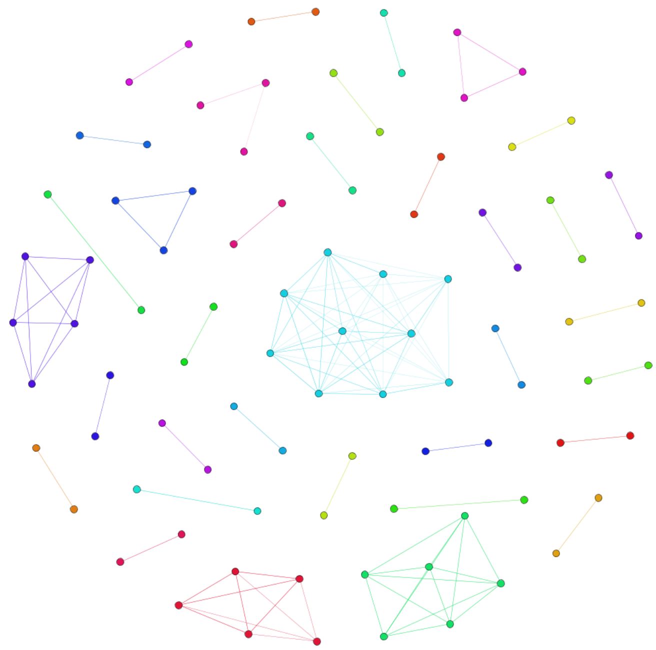


Figure 5: Overfitting cluster produced by improved MajorClust

refine cluster phase and is described as follows. First, we represent each cluster as a single node. We maintain the start and end time of the overall event log for each cluster to be considered for the anomaly score calculation. The frequency of an event occurring in all nodes in the cluster and updated $tf-idf$ is also counted.

To represent an event of a newly formed node, we need a string which reflects all events in a cluster. We can consider this as the longest common subsequence problem with many strings, or well-known as the multiple longest common subsequence (MLCS) technique. Solutions of this classical problem have been proposed in the recent literature [21, 22]. For each cluster $\mathbf{c}$, we will check for the most common substring occurring in the raw log. The terms "subsequence" and "substring" will be used interchangeably since the substring has the same meaning as a subsequence in our case, and we then describe MLCS concisely.

Let $l = l_1 l_2 \ \ldots \ l_p$ be a single log string in a particular cluster where the subscript number represents the index of a character in l, $s = s_{w_1} s_{w_2} \ \dots \ s_{w_q}$, p and q are their length respectively, s is called subsequence of l, $\mathrm{sub}(l, s)$, if it meets the following:

$$\mathrm{sub}(l, s) = \begin{cases} 1 \leq v \leq q & \text{if } 1 \leq w_v \leq p \\ 1 \leq t < u \leq q & \text{if } w_t < w_u \end{cases} \tag{8}$$

Furthermore, let $L = \{l_1, l_2, \ \dots \ , l_o\}$ be a set of raw events in the specific $\mathbf{c}$, o is total number of logs within each $\mathbf{c}$, multiple longest common subsequence for set L is a sequence of s if and only if (i) s is the subsequence of l_i for $1 \leq i \leq o$ and (ii) s is the longest one satisfying (i). We then apply MLCS to all clusters to find the longest substring of L.

After that, we run the MajorClust algorithm again using current graph composition. The result of the *refine cluster* phase is given in Figure 6. After this phase is complete, we convert the generated graph back to its original nodes representation since we still maintain the initial vertices of the formed new node. This technique will provide us with a better clustering result as shown in Figure 7. The output of this phase is a complete graph where each node has its own cluster label property.

It should be noted that the algorithm converges smoothly. The reasons are two-fold. First, the clustering algorithm works on a node by node basis. When a node is processed, the node is flagged, and the algorithm moves to the next node and so on. Second, the proposed improvement of MajorClust will ensure a node to have the same cluster as its heaviest edge neighbor. Using these two conditions, the algorithm will smoothly converge without any oscillation.

2.3. Anomaly detection of possible attacks

The nature of an anomaly is that it is different from other clusters [23]. Ideally, it will have a fewer members than others, but this assumption is not always true. Based on our extensive experiments and analysis on the public datasets from Security Repository (SecRepo) [24], the anomaly is the biggest cluster which has many nodes and the smallest cluster as well. In the context

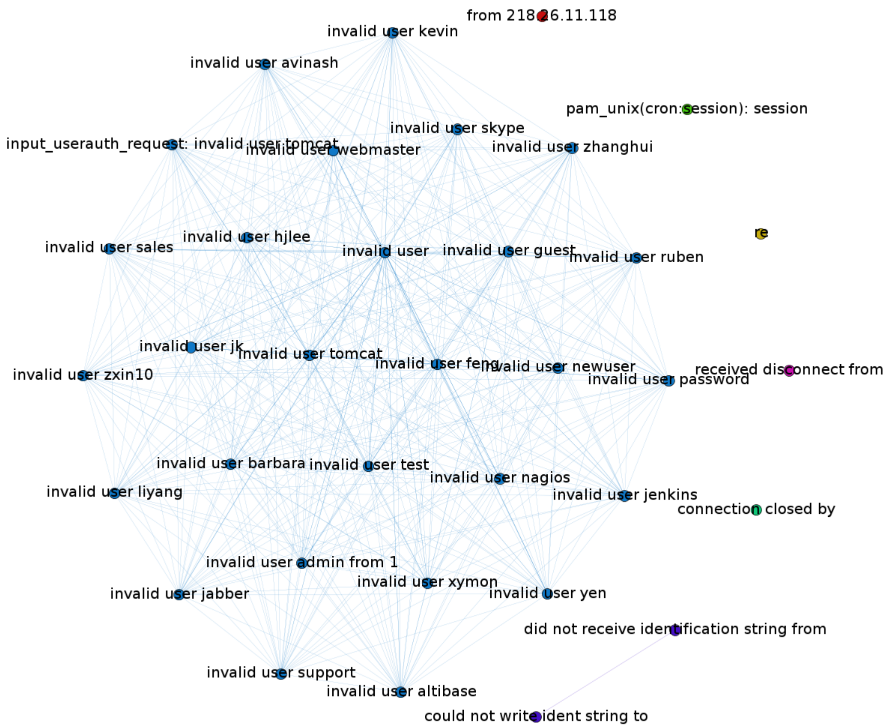


Figure 6: The result of *refine cluster* phase

of the access control log, the anomaly is highly correlated with total attacks or failed authorization of login attempts. The proposed technique will assist the forensic investigator to examine the real-life log that usually has a very large size.

One approach to discover the anomaly from the clustering result is to check the number of cluster members. If the number is below the given threshold, then it can be decided that there is an outlier in the cluster [25]. However, this procedure is not suitable for the access control case since the anomaly can occur with a large number of members in a cluster. Thus, we develop our own score to determine whether or not a cluster is an anomaly.

To achieve good outlier detection results, we define some parameters to calculate the anomaly score in every cluster, i.e., total event frequencies, total

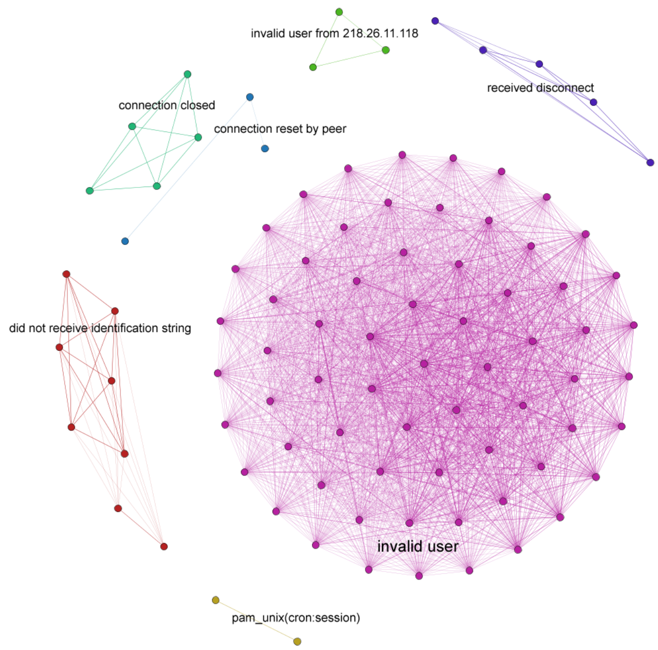


Figure 7: The final clustering of improved MajorClust algorithm

nodes, total members, and the inter-arrival rate of the event time. Note that the anomaly score measurement is conducted after the *refine cluster* phase and before it is converted back to the original graph representation.

We define the anomaly score as the ratio of event frequency per cluster to the total frequency in the graph. It is then multiplied by the inter-arrival rate between the first and the last events in a cluster. The high or low event frequency is likely to be an anomaly because the legal event will have a normal frequency. Furthermore, a very high or a very low inter-arrival rate can also be regarded as an anomaly. The event frequency and inter-arrival time between events are mainly used to detect SSH brute-force attacks. For example, some research include the event frequency [9] and inter-arrival rate of the events [26] as a part of the score to detect the malicious SSH activities. Both variables are considered in [10] as well.

First, we need the mean of frequency, μ_f, to characterize the number of members in a cluster c_i and define it as:

$$\mu_f = \frac{\sum_{i=1}^{n} f_i}{\sum_{i=1}^{n} \mathcal{N}_i} \tag{9}$$

where f_i and $\mathcal{N}_i$ are frequency and total nodes per vertex from the *refine cluster* phase, respectively. In addition, n is the total member of vertices in the cluster c_i. The anomaly will occur when there are so many events occurring in the adjacent time while the normal event shows the opposite behavior. Formally, the inter-arrival rate of all events in a cluster, $\mathbb{I}$, is formulated as follows:

$$\mathbb{I} = \frac{\sum_{i=1}^{n} f_i}{t_n - t_1} \tag{10}$$

where t_1 is the first time an event occurred, t_n is the last one, and both of them are measured in seconds. When the denominator produces a zero value, it means that there are several events in a second, and we set $t_n - t_1$ with very small value to show that its arrival rate is very high in the short time frame.

Furthermore, the anomaly score per cluster a is calculated based on the formula below:

$$a = \frac{\mathbb{M} \times \mu_f}{\sum_{i=1}^{z} \mathbb{N}_i \times \sum_{i=1}^{z} \mathbb{F}_i} \times \mathbb{I} \tag{11}$$

where $\mathbb{M}$ is total members in $\mathbf{c}$, $\mathbb{N}$ is total nodes in a cluster, $\mathbb{F}$ is total frequencies in $\mathbf{c}$, and z is total clusters in the analyzed log data. We refer to Equation (9) and (10) for μ_f and $\mathbb{I}$, respectively.

2.3.1. Estimation of anomaly threshold

To decide whether or not a cluster is an anomaly, we estimate a threshold to provide a recommendation for the forensic investigator or security analyst. We introduce an estimation for the anomaly parameter. There are three types of activities recorded in the log file: low intensity, normal, and high intensity [10, 27]. Low intensity refers to a sporadic attack while high intensity usually related to a brute-force attacks. In addition, the normal activities will produce a non-suspicious log.

First, the anomaly score that is calculated before is normalized to [0,1] as below. This will make a number of scores to fall in the same range.

$$a'_i = \frac{a_i - \min(A)}{\max(A) - \min(A)} \tag{12}$$

where a_i is an anomaly score in a cluster i, A is a set of anomaly scores for all clusters, and a'_i is a normalized score of a_i.

Next, we fit this normalized score to a quadratic equation since this equation fits the characteristics of the dataset where low and high intensity event will be defined as anomalies. The quadratic equation fits a' as axis to [0,1] and a'' as ordinate to [0,1]. The higher a'' score, the more normal the events.

$$a''_i = a'^2_i + 4a'_i - 4 \tag{13}$$

Subsequently, we have a second normalization step for anomaly score a'' to range [-1,1] where 0 is the threshold for the anomaly decision. The final anomaly score per cluster α_i is depicted below where $p = -1$ and $q = 1$ as the range boundaries.

$$\alpha_i = p + \frac{(a''_i - \min(A'))(q - p)}{\max(A') - \min(A')} \tag{14}$$

If the α_i score is less than 0, then a cluster i is set as an anomaly and vice versa. Thus, the user is not required to enter the parameter to calculate the anomaly score.

2.4. Visualization of access control anomaly

In order to help the investigators in analyzing and getting a better understanding of the security log, there are some methods available for log visualization. For example, Takada and Koike created Tudumi, a visualization tool for auditing `syslog`, `wtmp`, and `sulog` based on layered concentric disks [28]. For each layer, there were some notations such as spheres and cubes to represent the user and his activities. The treemaps model to display clustering result from the Simple Log file Clustering Tool (SLCT) [12] of the event log was introduced and named LogView [29]. Another approach was using parallel coordinates to plot several logs, e.g., network, database, and `syslog`.

Elvis (Extensible Log Visualization) is a recent implementation to display an Apache log, `syslog`, and `auth.log` [30]. This technique was based on a custom organization (categorical and geographical) and log augmentation. Visual Filter enabled the security analyst to inspect the whole logs, creates a filter visually, performs navigation, and sub-selects the part of the log interactively [31]. In addition, Trethowen et al. introduced VisRAID to visualize remote access logs, especially for intrusion detection purposes and focused on timeline visualization [32].

However, existing methods do not apply graph visualization to the assistance tool for log analysis. Naturally, since the clustering is based on a graph, the proposed visualization also heavily relies on this model. Therefore, we propose analysis and anomaly display of the authentication log based on graph visualization. We categorize the visualization into two types: static and dynamic. The static model displays the graph after the analysis is complete while the dynamic one provides live visualization when the analysis is running.

We use Gephi for the graph visualization for both types [33]. We then exploit the streaming server plugin [34] and its Python client implementation [35] to support a dynamic model. In the static mode, the visualization is displayed after the clustering and anomaly detection process is finished. The output of the forensic analysis is a graph file, i.e., dot file and then it is exported to Gephi. To get the intuitive display, we use two steps of the graph layout algorithm: first Force Atlas 2 [36] and then Fruchterman Reingold [37]. These two algorithms have been natively integrated into Gephi.

Force Atlas 2 produces clustered nodes, but there are still many overlaps between them, Fruchterman Reingold will remove these obstacles, and provide a clearer layout. While in the dynamic mode, the visualization follows the analysis process starting from creating nodes, edges, graph clustering, refining the cluster, and anomaly detection. The result is clean and concise since the procedure has removed unnecessary edges that connect one cluster to the other one. For example, Figures 2 to 7 are produced in the static mode.

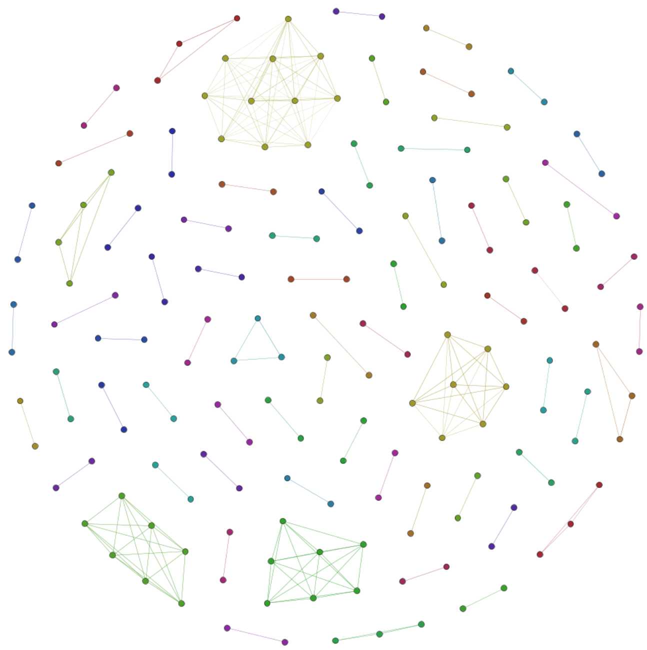


Figure 8: Initial clustering result of the first day in the dataset (November 30, 2014)

3. Experimental results and discussions

3.1. Functionality testing for SecRepo dataset

The dataset for the experiments is taken from the public and open Security Repository (SecRepo). It includes 86,839 lines of `auth.log` taken from November 30 to December 31, 2014, and mainly contains failed SSH login attempts [24]. This kind of attack will be flagged as an anomaly in the whole log file. We utilize NetworkX to create and manipulate the graph [38], Gephi as graph visualization tools [33], a Gephi streaming server plugin [34] and Gephi streaming client [35] to dynamically draw a graph to Gephi, and Natural Language Toolkit (NLTK) to provide English corpus for log text preprocessing, especially in providing stopwords [39]. We utilize the Python programming language version 2.7 to assemble all of these libraries and provide this log forensic tool.

Figure 8 shows the initial clustering result of the first day in the dataset

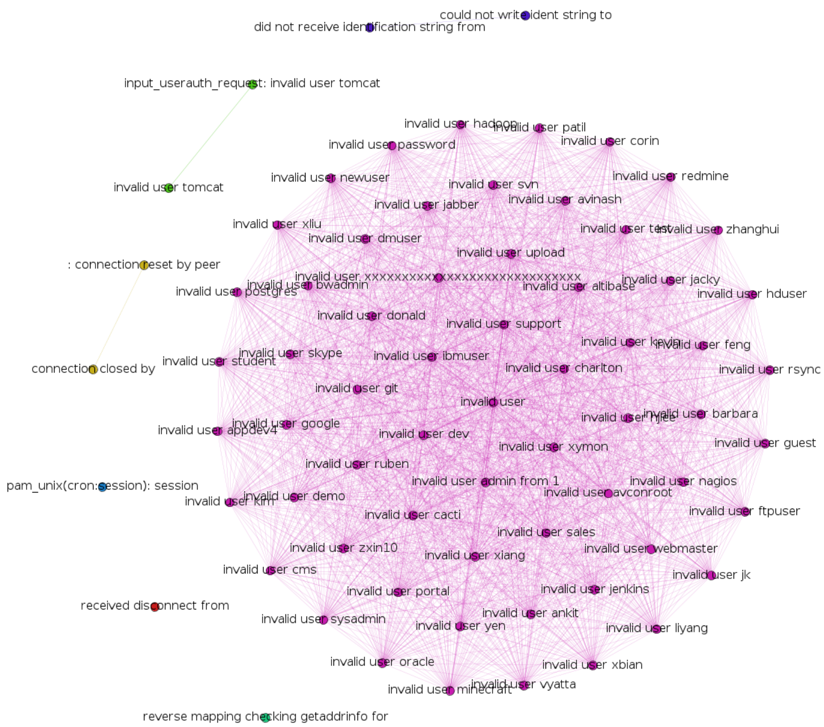


Figure 9: The result of *refine cluster* phase from Figure 8

(November 30, 2014) while Figure 9 illustrates the *refine cluster* phase. There are too many small clusters in the initial step and it is fixed in the next phase. The anomaly detection result is depicted in Figure 10 where the red vertices represent the anomaly and the green represents the normal events. This figure is completed with the label of an event in the access control log. The outlier is measured when a cluster is on the *refine cluster* phase based on the estimated threshold.

We can see in Figure 10 that the detected anomaly is an `invalid user` event as shown in the largest cluster. There are small clusters with similar events such as `invalid user tomcat`, `invalid user cms`, `invalid user dev`, and `invalid user google`. These clusters create their own cluster since the nodes inside them have the neighbor with the heaviest weight and do not follow

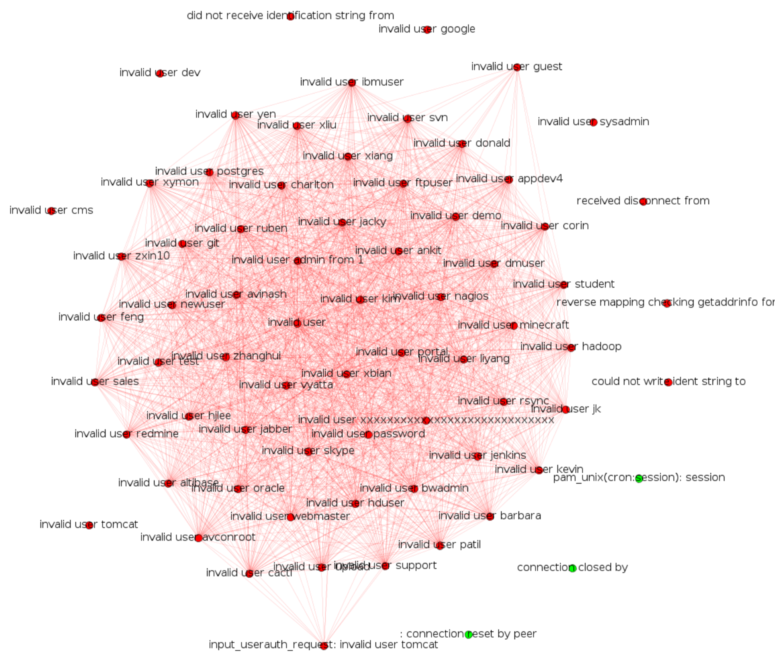


Figure 10: Anomaly detection in *refine cluster* phase

the bigger cluster, which has less cosine similarity.

In addition, the `pam_unix(cron:session) session` is a daemon event run in the background so it is successfully categorized as normal. The other two nodes in green, i.e., `connection close by` and `connection reset by peer` is a false positive since these events will only happen when there are failed authorizations. The calculation of the accuracy and performance of the proposed method will be explained later in this section.

Figure 11 is the full version of Figure 10 which is converted back to its initial graph representation where it has 688 rows of the event log and is modeled into 173 nodes and 9,694 edges. To produce a good display of anomaly detection, these visualizations are generated after two runs of the layout algorithm, i.e., Force Atlas 2, then followed by Fruchterman Reingold, and successfully implemented in the Gephi graph editor for both static and dynamic modes.

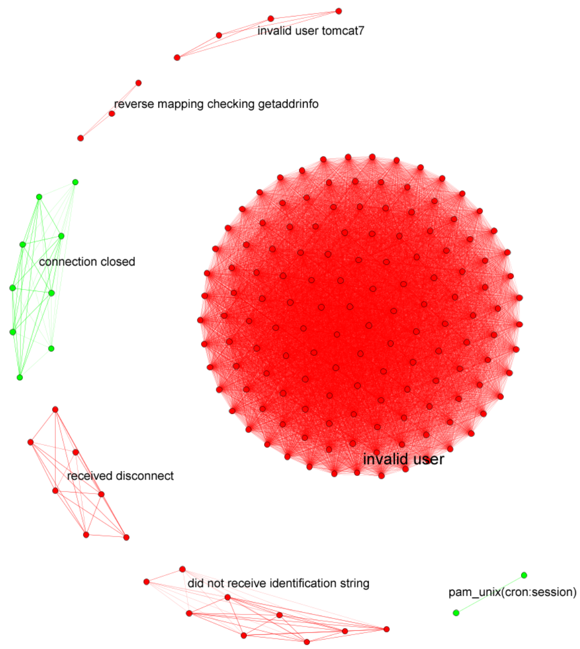


Figure 11: Final result of the forensic analysis in the first day of authentication log (November 30, 2014)

3.2. Evaluation metrics

Furthermore, we use some standard measurements to evaluate the performance of the proposed method such as true positive (tp), true negative (tn), false positive (fp), false negative (fn), sensitivity, specificity, and accuracy. The sensitivity and specificity are defined in Equation (15) and (16), respectively [20]:

$$Sensitivity = \frac{tp}{tp + fn} \tag{15}$$

$$Specificity = \frac{tn}{fp + tn} \tag{16}$$

While sensitivity is focused on measuring the improved MajorClust's ability to detect a normal event, the specificity and accuracy will provide an overall measure since it also includes true negative (tn) as denoted below:

$$Accuracy = \frac{tp + tn}{tp + fp + fn + tn} \tag{17}$$

In this case, true negative refers to detected *anomaly* record is also defined as *anomaly* in the dataset. Before calculating tp, tn, fp, and fn, we label the dataset with *normal* and *anomaly* then compare it to the proposed method's decision.

3.3. Comparison with existing methods

To compare the robustness of the improved MajorClust and proposed anomaly score, we evaluate other methods to the same testing dataset. We run the Simple Log Clustering Tool (`slct`) version 0.05 [12] and `LogCluster` version 0.03 [40] as the clustering-based anomaly detection in event logs. In addition, we also compare the proposed method with two rule-based anomaly detections, i.e., `OSSEC` version 2.8.3 [2, 3, 4] and `swatch` version 3.2.3 [41]. These two applications can be turned into forensic analysis tools since they were initially designed to monitor real-time log file. We also compare the proposed method with standard MajorClust and improved MajorClust without *refine cluster* phase. They will be combined with the proposed outlier detection mechanism since they do not have one.

3.3.1. Parameter tuning for sclt and LogCluster

To objectively compare those methods, we have trained other tools' parameter to get their best accuracy. Therefore, the dataset was divided into two parts. The first part is November 30 to December 14 as the training set and the second one is December 15 to 31 as the testing set. The training set is used for parameter tuning while the testing set is for performance comparison as presented in Subsection 3.3.2.

The parameter *-support* or *-s* in slct and *--support* in LogCluster have been tuned from 10 to 100. Then we checked which value generates the best accuracy. Based on the experiments, parameter *-s* for `slct` is set to 100 which means that

Table 1: Parameter tuning -*s* for **slct** and --*support* for **LogCluster**

Value of -*s* or --*support*	Accuracy for **slct** (%)	Accuracy for **LogCluster**
10	2.38	5.06
20	2.32	3.47
30	3.04	3.56
40	2.99	4.82
50	3.43	6.23
60	2.64	6.00
70	3.34	7.65
80	3.26	8.28
90	3.36	9.51
100	**3.62**	**10.23**

the threshold for the number of cluster member is 100 lines. As a consequence, log lines that do not belong to any cluster will be defined as an anomaly. The similar parameter is applicable to `LogCluster` and we set the option --*support* to 100 since it produces the best accuracy. The result of parameter tuning for `slct` and `LogCluster` is presented in Table 1.

The rules used in `OSSEC` and `swatch` are based on regular expressions. Since the `swatch` tool does not provide any standard rules, we set them by adding three most frequent malicious strings in the event log. They are (`[iI]nvalid [uU]ser`, `Did not receive identification string`, and `reverse mapping`). On the other hand, the OSSEC tool has provided the set of rules in an XML file. We followed these default rules.

3.3.2. Comparison results

The comparison of the proposed technique and the state of the art is given in Table 2. As shown in Table 2, the best sensitivity value was achieved by `swatch`. This indicates that it has a very good capability to detect the *normal* activities in the log since we manually configure the regular expression rules that commonly occurred in the testing set. The proposed method achieved the best

Table 2: Comparison of proposed technique and the other methods

Methods	Sensitivity (%)	Specificity (%)	Accuracy (%)
Proposed method	70.59	**82.21**	**83.14**
Proposed method without *refine cluster* phase	88.24	50.25	52.43
Standard MajorClust [17]	0.00	78.08	73.61
`slct` [12]	87.50	4.88	13.54
`LogCluster` [40]	73.21	17.94	23.73
`OSSEC` [2]	25.11	30.12	29.59
`swatch` [41]	**100.00**	54.13	58.94

sensitivity and specificity rate of 70.59% and 82.21%, respectively. On the other hand, the other techniques produce lower specificity. The improved MajorClust is able to provide highest accuracy value (83.14%) while the others do not. The difference in accuracy is significant which indicates that existing methods are unable to properly detect the suspicious logs in the testing data.

We can see that the standard MajorClust without modifications does not work well and even it gives zero sensitivity. On the other hand, the *refine cluster* phase is hardly needed to improve the accuracy. Note that these two methods use the proposed anomaly score estimation.

The drawback of previous clustering-based anomaly detection is the assumption that the outlier is the data residing on a small cluster. This idea does not work in the authentication log with a high number of attacks since these violations produce many records and can create the largest cluster in the analysis. Moreover, the rule-based anomaly detections contribute to the poor performance of anomalous discovery when the supplied regular expressions do not suit with the case. The default rule should be updated regularly by the network analysts in order to make sure all security breaches are detected. However, it can produce good true positive rates (sensitivity).

To supply the forensic investigator with more insight, we also enumerate the most frequent events when processing the `auth.log` file as presented in Table

Table 3: The most frequent events in the authentication log (testing dataset)

No	Event	Frequency
1	`Invalid user`	12,743
2	`Received disconnect`	11,464
3	`pam_unix(cron:session)`	2,708
4	`IP does not map back to the address`	1,328
5	`Reverse mapping checking getaddrinfo failed`	1,216

3. The `invalid user` shows that there is an attempt from an unregistered user to login to the server. The event `received disconnect` means that an error occurred in the authentication process. The attacker is usually trying to conduct SSH brute force and the server decides to disconnect the connection.

There are warning messages such as `IP does not map back to the address` and `reverse mapping checking getaddrinfo failed` in the event. The system records these two events because the client's reverse DNS record does not match with the hostname used to identify the client's identity. Event `pam_unix(cron:session)` means that the daemon activity is generated from the default configuration of Linux operating system. This record is written when the system checks the authentication mechanism and it happens every hour by default. From this analysis, we can infer that there are so many attempts to violate the access control, especially SSH, in the server.

3.4. Experiment on the Kippo log

In this section, we present one more experiment to another dataset, i.e., Kippo log, to prove that the proposed method works well to another dataset with another attack. Kippo is an SSH honeypot and created in Python [42]. We install and configure Kippo on DigitalOcean droplet with public IP address so everyone including real attackers and botnet can reach this server. All the activities were recorded from 14 to 20 February 2017.

Beside the attack from the internet, we add one more activity that is usually done by the attacker before attempting penetration. The attack is a part of reconnaissance step to get banner or fingerprint of the SSH server version using the telnet application. This experiment shows that the proposed method performs well with 52.89% sensitivity, 92.77% specificity, and 87.35% accuracy.

4. Conclusion and future works

We propose an improved MajorClust algorithm to cluster the authentication log as the basis of anomaly detection of suspicious activities related to access control violations. The experimental results show that the proposed method is able to provide assistance to the forensic investigator, security analyst, or system administrator in inspecting and visualizing the log file. In the experiment, it achieves 70.59% of sensitivity, 82.21% of specificity, and 83.14% of accuracy.

In future, we would like to improve the similarity measure between two log records using semantic analysis since it considers the meaning, context, and linguistic aspect while $tf-idf$ only calculates the frequency of terms. This measurement is expected to increase the accuracy of the anomaly detection phase. The clustering of the log file is still an open challenge for good performance of the analysis since the improved MajorClust, in some circumstances, failed to produce a precise cluster. In addition, the proposed method can be deployed to other types of event logs such as syslog, web server log, web proxy log, and many more.

Since the log data has been converted to graph data structure, we will add tamper detection using graph watermarks in the future. This method works by adding a subgraph to the main graph completed with a graph key and the instance's key [43]. The instance can be another process which saves the log files and it can reside in the same server as event log or in other servers. Watermarks guarantee the integrity of the log files and also provides a leak detection, so the suspected instance will be easily detected.

It is also worth noting that the runtime of the authentication log forensic analysis can still be improved. Some research in the digital investigation area has considered this problem [44]. We plan to implement the parallel version of the proposed method to increase the processing time so that the authorities will receive the result and report it immediately. This is one of our priorities since we believe that the evidence not only originates from a single host but can also be obtained from a cluster environment.

Acknowledgments

This work is supported by the Indonesia Lecturer Scholarship (BUDI) from Indonesia Endowment Fund for Education (LPDP). This scholarship is a collaboration between the Ministry of Research, Technology, and Higher Education with the Ministry of Finance, Republic of Indonesia. We also thank two anonymous reviewers who have provided the constructive comments to this manuscript.

References

[1] K. Geisshirt, Pluggable authentication modules, Packt Publishing, Birmingham, UK, 2007.

[2] OSSEC, Open Source HIDS SECurity.
URL `http://ossec.github.io/`

[3] D. B. Cid, Log analysis for intrusion detection, Tech. rep. (2009).

[4] A. Hay, D. B. Cid, R. Bray, OSSEC host-based intrusion detection guide, Syngress Publishing, 2008.

[5] M. Sato, T. Yamauchi, VMM-based log-tampering and loss detection scheme, Journal of Internet Technology 13 (4) (2012) 655–666.

[6] D. Basin, P. Schaller, M. Schläpfer, Logging and log analysis, in: Applied information security, Springer-Verlag Berlin Heidelberg, 2011, pp. 69–80.

[7] A. Abdou, D. Barrera, P. C. V. Oorschot, What lies beneath? Analyzing automated SSH bruteforce attacks, in: Proceedings of the 8th International Conference on Passwords, 2015, pp. 72–91.

[8] J. L. Thames, R. Abler, D. Keeling, A distributed active response architecture for preventing SSH dictionary attacks, in: Proceedings of the 2008 IEEE SoutheastCon, 2008, pp. 84–89.

[9] Y.-N. Su, Developing the upgrade detection and defense system of SSH dictionary-attack for multi-platform environment, iBusiness 03 (01) (2011) 65–70.

[10] M. Javed, V. Paxson, Detecting stealthy, distributed SSH brute-forcing, in: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, 2013, pp. 85–96.

[11] N. L. Beebe, J. G. Clark, Digital forensic text string searching: Improving information retrieval effectiveness by thematically clustering search results, Digital Investigation 4 (2007) 49–54.

[12] R. Vaarandi, A data clustering algorithm for mining patterns from event logs, in: Proceedings of the 2003 IEEE Workshop on IP Operations and Management, 2003, pp. 119–126.

[13] R. Vaarandi, Mining event logs with SLCT and LogHound, Proceedings of the IEEE Network Operations and Management Symposium (2008) 1071–1074.

[14] M. E. Celebi, H. A. Kingravi, P. A. Vela, A comparative study of efficient initialization methods for the k-means clustering algorithm, Expert Systems with Applications 40 (1) (2013) 200–210.

[15] F. Murtagh, P. Contreras, Algorithms for hierarchical clustering: An overview, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2 (1) (2012) 86–97.

[16] H.-P. Kriegel, P. Kröger, J. Sander, A. Zimek, Density-based clustering, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1 (3) (2011) 231–240.

[17] B. Stein, O. Niggemann, On the nature of structure and its identification, in: Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science, 1999, pp. 122–134.

[18] E. Levner, D. Pinto, P. Rosso, D. Alcaide, R. R. K. Sharma, Fuzzifying clustering algorithms: the case study of majorclust, in: Proceedings of the 6th Mexican International Conference on Artificial Intelligence, 2007, pp. 821–830.

[19] O. Niggemann, V. Lohweg, T. Tack, A probabilistic majorclust variant for the clustering of near-homogeneous graphs, in: Proceedings of the 33rd Annual German Conference on Artificial Intelligence, 2010, pp. 184–194.

[20] C. D. Manning, P. Raghavan, H. Schütze, An introduction to information retrieval, Cambridge University Press, 2009.

[21] J. Yang, Y. Xu, G. Sun, Y. Shang, A new progressive algorithm for a multiple longest common subsequences problem and its efficient parallelization, IEEE Transactions on Parallel and Distributed Systems 24 (5) (2013) 862–870.

[22] J. Yang, Y. Xu, Y. Shang, G. Chen, A space-bounded anytime algorithm for the multiple longest common subsequence problem, IEEE Transactions on Knowledge and Data Engineering 26 (11) (2014) 2599–2609.

[23] C. C. Aggarwal, Outlier analysis, Springer, 2013.

[24] M. Sconzo, SecRepo.com: Security data samples repository (2016).
URL `http://www.secrepo.com/`

[25] A. Loureiro, L. Torgo, C. Soares, Outlier detection using clustering methods: a data cleaning application, in: Proceedings of KDNet Symposium on Knowledge-based systems for the Public Sector, 2004.

[26] A. Satoh, Y. Nakamura, T. Ikenaga, A flow-based detection method for stealthy dictionary attacks against Secure Shell, Journal of Information Security and Applications 21 (2015) 31–41.

[27] Y. Zhong, H. Yamaki, H. Takakura, A grid-based clustering for low-overhead anomaly intrusion detection, in: Proceedings of the 2011 5th International Conference on Network and System Security, 2011, pp. 17–24.

[28] T. Takada, H. Koike, Tudumi: Information visualization system for monitoring and auditing computer logs, in: Proceedings of the International Conference on Information Visualisation, 2002, pp. 570–576.

[29] A. Makanju, S. Brooks, A. N. Zincir-Heywood, E. E. Milios, LogView: Visualizing event log clusters, in: Proceedings of the 6th Annual Conference on Privacy, Security and Trust, 2008, pp. 99–108.

[30] C. Humphries, N. Prigent, C. Bidan, F. Majorczyk, ELVIS: extensible log visualization, in: Proceedings of the 10th Workshop on Visualization for Cyber Security, 2013, pp. 9–16.

[31] J.-E. Stange, M. Dörk, J. Landstorfer, R. Wettach, in: Proceedings of the 11th Workshop on Visualization for Cyber Security, 2014, pp. 41–48.

[32] L. Trethowen, C. Anslow, S. Marshall, I. Welch, VisRAID: visualizing remote access for intrusion detection, in: Proceedings of the 20th Australasian Conference, 2015, pp. 289–306.

[33] M. Bastian, S. Heymann, M. Jacomy, Gephi: an open source software for exploring and manipulating networks, in: Proceedings of International AAAI Conference on Web and Social Media, 2009, pp. 361–362.

[34] A. Panisson, Gephi streaming plugin (2013).
URL `https://marketplace.gephi.org/plugin/graph-streaming/`

[35] A. Panisson, Python client for graph streaming on Gephi (2014).
URL `https://github.com/panisson/pygephi_graphstreaming`

[36] M. Jacomy, T. Venturini, S. Heymann, M. Bastian, ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software, PLoS ONE 9 (6) (2014) 1–12.

[37] T. M. J. Fruchterman, E. M. Reingold, Graph Drawing by Force-directed Placement, Software-Practice and Experience 21 (11) (1991) 1129–1164.

[38] A. Hagberg, D. Schult, P. Swart, Exploring network structure, dynamics, and function using NetworkX, in: Proceedings of the 7th Python in Science Conference, 2008, pp. 11–15.

[39] S. Bird, E. Klein, E. Loper, Natural language processing with Python, O'Reilly Media, Inc., 2009.

[40] R. Vaarandi, M. Pihelgas, LogCluster - a data clustering and pattern mining algorithm for event logs, in: Proceedings of the 11th International Conference on Network and Service Management, 2015, pp. 1–7.

[41] T. Atkinds, swatch: Simple log watcher (2015).
URL `https://sourceforge.net/projects/swatch/`

[42] Kippo, SSH honeypot.
URL `https://github.com/desaster/kippo`

[43] X. Zhao, Q. Liu, H. Zheng, B. Y. Zhao, Towards graph watermarks, in: Proceedings of the 2015 ACM on Conference on Online Social Networks, 2015, pp. 101–112.

[44] D. Quick, K. K. R. Choo, Impacts of increasing volume of digital forensic data: A survey and future research challenges, Digital Investigation 11 (4) (2014) 273–294.