# Identifying Irregularities in Security Event Logs through an Object-based Chi-squared Test of Independence

Simon Parkinson[a,*], Saad Khan[a]

[a]*Department of Informatics, School of Computing and Engineering, University of Huddersfield, Queensgate, Huddersfield, HD1 3DH, UK*

**Abstract**

A novel technique for identifying irregular event log entries is presented in this paper along with the implementation in a Microsoft Windows-based environment. The motivation behind this research is to identify irregular activity in a system whilst minimising any requirement on expert knowledge, in addition to saving investigative time and computing resources. As the developed solution utilises the standard Microsoft format for event logs, it can work with both live systems, as well as events extracted and stored for off-site analysis. The solution consists of two major steps: first, convert the event logs into objects-based model and second, perform statistical analysis using the Chi-squared ($\chi^2$) test of independence and classify mean $\chi^2$ values into discrete categories using Jenks natural breaks method. The event logs entries, which failed the test of dependence are considered as irregular events. It is also shown that the proposed solution poses an advantage over primitive frequency analysis methods as it uses object relationships among event log entries to determine irregularities for locating anomalous activities. Empirical analysis of the solution is performed using event logs data from 20 machines and shows promising results by correctly identifying irregular events. Further experimental analysis involving the insertion of synthetic irregular events results in an average accuracy of 85%.

*Keywords:* Chi-square ($\chi^2$), Test of independence, Jenks natural break, Frequency analysis, Irregular events, Automated expert analysis

## 1. Introduction

An event log contains a sequence of discrete events that are automatically constructed by a computer's operating system, running software services, and user created applications. In general, event logs have information related to security, performance and error events. An event log stores data for retrieval at a later date by a security professional or use by an automated security system. Some applications and services have their own bespoke mechanism of logging, form example through the use of bespoke data formats stored in Comma Separated Value (CSV) or eXtensible Markup Language (XML) format. It is also common that the operating system itself provides a centralised mechanism for storing and reporting event logs. This centralised mechanism makes it easier for a user to examine and search for relevant content. In this paper we focus on processing the Microsoft event logging mechanism, although the develop techniques are general and can be applied elsewhere to different logging mechanisms.

An event log stores information that can be essential in performing system diagnostics and security audits. By default, the system will log event information which is believed to be of relevance, although in the Microsoft system event logging can be enabled for many additional security configuration events that may be of further benefit during an auditing procedure. For example, enabling event logging for changing of file

*Corresponding author
*Email address:* s.parkinson@hud.ac.uk (Simon Parkinson)

system permissions and assigning users to groups. Another useful feature of the logging mechanism is that the occurrence of an event is recorded in terms of objects (name-value pairs). These objects correspond to timed entities and provide comprehensive details of the event, such as username, permissions and timestamp, which is useful for analysis.

Many experienced administrative and security practitioners are aware of what information is available in the event log and will have knowledge and expertise of how to search for information of interest. However, this process can be troublesome during time-critical processing. Performing a manual audit of the event log to identify or reject investigative hypothesise is time-consuming, even with experience and expert knowledge. Another challenge exists for users with less experience and expertise as access to expertise becomes expensive through the use of consultancy services and automation software. In some instance the event log might store information critical to the security and usability of an individual's computer, but without in-depth knowledge and expertise the user is unable to identify and interpret this information [1].

The aforementioned challenges motivates further research into the identification and extraction of useful information held within the event log. There are techniques available in the public domain that perform an automated search of an event log and identify events of interest by comparing them against a known knowledge-base. Although such approaches are only useful for identifying events that are known to be of interest, they are not capable of identifying events that are potentially of significance in terms of security and usability due to their low frequency of occurrence, as well as their irregular involvement with the operating system's underlying objects. Here the set of underlying objects could include a user and a collection of directories. An irregularity in terms of event log entries could be that a user has been granted a higher than normal level of access permissions on a directory, which is reported within the event log. The automated identification of such irregular events would allow the user to determine suspicious activities in a resourceful and efficient manner.

The aim of this paper is to introduce a technique for automatically identifying irregular event log entries. This paper builds on previous work where researchers have developed an unsupervised technique to identify irregular file system permissions using statistical analysis [2]. This requires modelling the event log in a way that allows us to use a statistical test of independence between an event's objects and their distribution among all events. The research hypothesis addressed in this paper is: using statistical methods on event logs can assist in identifying event log entries of interest without programatically encoding knowledge. The primary contributions presented in this paper are:

- **A novel technique for identifying irregular event log entries**. This automated technique takes an *object-centred* modelling perspective to event log entries. Following this, a statistical analysis ($\chi^2$) technique is used to identify event log entries which fail a test of dependence. Empirical analysis is then provided to evaluate the effect of event log complexity, and the frequency distribution of event log objects.

- **A novel software tool for interacting with Microsoft's event log mechanism**. The produced software tool enables users with limited experience and expertise of the Microsoft event logging mechanism to quickly identify irregular, and potentially interesting, event log entries. Furthermore, the software tool has the ability to provide an automated solution to find interesting events, and therefore reducing investigation time required by an expert.

- **Empirical analysis and case study**. The significance of the proposed techniques and tool is then evaluated through the use of 20 real event logs to determine overall performance and usefulness. This analysis is then supplemented by inserting synthetic irregular security event, and the technique results in an 85% average accuracy.

The remainder of the paper is organised as follows: Section 2 contains a survey of related work and discusses state-of-the-art in event log auditing. Section 3 presents the modelling of an event log into a suitable form for further processing. Section 4 presents a simplistic frequency analysis to find event log types which appear both frequently and infrequently. This then leads on to Section 5 where the development of a object-based model that is subsequently used alongside a statistical test of independence and is used to

determine irregular event log entries. Examples and empirical analysis regarding performance and accuracy are also presented. Section 7.2 presents a large-scale evaluation both on synthetic and real event log systems to establish the performance and accuracy of the proposed technique.

## 2. Related Work

An event log is a record of any significant or erroneous activity by a user or program. It usually contains a complete set of information (such as date, time, user account, machine, description, type, etc.) that can help in understanding that specific occurrence in the system [3]. Event logs can be generated by users, applications and the system itself. As the event logs can often be produced in high frequency (dependent on configuration), and their total number expands over time, the major concern is their proper management as well as in extracting useful and actionable information. Many studies have been undertaken to address these challenges. In one study, the concepts of *event patterns*, *event filtering*, *event aggregation* and *causal event histories* are presented to provide an overview of *Complex Event Processing* [4] in a distributed system.

The event patterns are the rules, which are used in event filtering mechanism to gather relevant or required events logs. Their relation can be described in terms of time, dependency or mutual exclusion. The related event logs are used as an input of event aggregation system, which outputs a map of newly generated (abstract) higher level events and their causal relationships. The causal event histories defines the relationship(s) of one event to another with respect to the activity's semantics. In another study [5], event logs are analysed using the $\alpha$ algorithm. The technique was successfully able to rediscover workflow process models of two real-time applications (health-care and judicial system) without any knowledge, alongside the presence of unrelated events (noise). The extracted information is then converted in the form of SWF-net (part of the Flash language) to make it visually more useful for the end user. A similar study [6] focuses on the generation of event logs through social networks. They extend process mining, filtering and clustering techniques to retrieve information from the event logs, build an ordered sets using time as a metric, and then present the results in the form of Petri-nets.

As previously mentioned, event logs contain a large amounts of text data, which make analysing them a computationally challenging task. Primitive techniques do exist for debugging, searching, visualisation and characterisation of event logs, such as *sequential text clustering* and *Principle Atom Recognition In Sets* (PARIS) algorithms [7]. The approaches provide mechanisms to help perform structured analysis of an event log's contents. These mechanisms are effective; however, a large amount of investigative effort is still required to identify irregular events of interest.

The availabilities of large amounts of event logs can be turned into an advantage if pattern analysis is performed. One such approach is called frequent patterns mining [8], which can detect regularities and anomalies in the data. In general, these algorithms take an entire set of event logs and slices them into a time frame (e.g. 1 second). A specific combination of event logs is considered to be a frequent pattern if it is present in a number of time frames. A study [9] provides an efficient breath-first algorithm for mining frequent patterns in event logs. The authors claims that as the memory consumption of counting items in each slice is high, the algorithm only counts those items that do not observe the pattern. I.e filtering out those known to not be relevant. To further improve the performance, the algorithm considers a specific set of event data properties; number of all items and transactions, items that occur less than 10 times, and items that occur at least once per 1000 transactions.

In other studies aimed at developing techniques to further improve accuracy and performance, techniques such as (frequent pattern) *FP-growth* algorithm [10] and its variants such as *Parallel FP-Growth* [11] and *Balanced Parallel FP-Growth* [12] are proposed. The FP-Growth algorithm is claimed as one of the fastest algorithm to determine associations, correlations, and causal structures among the sets of items. There are also many other algorithms that improve upon various problems related to size, scalability and efficiency. Some of them are *FreeSpan* [13], graph-based approach called *gspan* [14] and tree-based approaches known as frequent pattern tree (FP-tree) [15], *CanTree* [16] and *CP-tree* [16]. The aim of these approaches is to combine the itemsets for identifying strong and frequent co-occurrences but they perform the task in a different structural forms, such as subgraphs, subtrees, or sublattices. In addition to finding correlations, frequent patterns help in numerous research frontiers like data indexing, classification and clustering. These

graph-based approaches which are used to identify strong relationships (strongly connected subgraphs) can also be used to identify irregular and poorly connected sub graphs through inverting the evaluation function. I.e., identifying those below a threshold rather than above.

Machine learning techniques have also be applied to analyse event logs. A tool called, *Decision Miner* [17] uses decision trees to produce possible decision paths within a Petri net model to identify the dependencies. The petri-net model is generated from event logs. The candidate paths are used as an input for already trained *J48 decision tree* classifier and *C4.5* algorithm, which outputs the final optimal decision. Another tool named *Little Thumb* [18] is capable of capturing the concurrent event logs using a heuristic process mining technique (by providing dependency/frequency table) and workflow petri-nets. Event log analysis can also be performed for finding anomalies in the system. A similar study [19] has extracted identifiers and state variables via console logging of over 24 million messages and used Principal Component Analysis (PCA)-based unsupervised learning algorithm for anomaly detection. A similar system called Magpie [20] extracts event logs traces across multiple systems of an e-commerce site and uses clustering algorithms to build a probabilistic model of user's request behaviour to detect anomalies. The machine learning algorithms can further be improved through utilising different supervised learning mechanisms, for example the stochastic gradient descent. It is shown [17] that the performance of *Perceptron*, *Adaline*, *k-Means*, *Support Vector Machines* and *Lasso* algorithms have significantly been boosted using stochastic gradient descent.

Forensic analysis can also be used to extract, collect, correlate and interpret the complex and high volume of event logging data, with a view to acquire investigative evidence. Current forensic techniques have become more automated and accurate in nature and provides better efficiency and performance [21]. For this purpose, a model checking approach have been proposed to perform the forensic analysis based on modal, temporal, linear and dynamic characteristics of the event logs [22]. The proposed algorithm formalises the information inside each event log into algebraic mathematical symbols and then a constructs a tree to convey all dependency relations. Another study presents a Principles-driven [23] forensic analysis technique, which emphasises on determining the cause and effect (context) of actions based on the event logging data. Authors claims that time is of key importance to improve the analysis process along with a multi-resolution view of the data, which considers multiple metrics (users-space, system/kernel-space, environment etc.) at any given instant. A patented piece work [24] provides a method to collectively analyse the heterogeneous event logs from multiple devices. First, security event logs are remotely gathered and normalized into a common schema. The authors cross-correlated the logs on the basis of pre-defined rules for producing meta-events, which are then reported via e-mail or telephone messages.

This section has demonstrated that a wealth of research has been performed into processing event logs to mine useful information, both in a security auditing and forensic context. However, many of the techniques utilise and process limited aspects of an event's details, particular those with a systematic and predicable structure. More specifically, type and time are the most commonly used aspects. In this work, we investigate processing an event's description to extract an object-centric model which can further enhance event log analysis, with a particular focus on identifying irregular events of interest.

## 3. Modelling

The structure of event logs changes depending on the underlying system and event cause. In this work, we consider the Microsoft event logging system, whereby event log entries are created by system services, as well as any software executing on the host system. The structure of an event entry is fixed; however, the structure and contents of the description depends upon what information the system programmer decided to include in the event. Some systems contain human readable content, whereas other contain information useful for debugging experts and requires expert knowledge to interpret. It is therefore necessary to have a generic mechanism to process content of the event logs. In this research, a generic approach is taken where the event description is assembled to contain a set of objects.

An event log, $E$, consists of a series of events ($E = \{e_1,e_2,\ldots,e_n\}$) where each individual event is a tuple, $e = \{T,I,O\}$, consisting of a timestamp ($T$), an event ID ($I$), and a set of objects denoting the event description $O = \{o_1,o_2,\ldots,o_n\}$. In the Microsoft environment, an event's description will often contain information dictated by the programmer and the event type. For example, error log entries for applications

```
Permissions on an object were changed.
     ID: log ID 4670
Subject:
     Security ID:    admin
     Account Name:   Bob
     Account Domain: AD
     Logon ID: 0x9B3EC
Object:
     Object Server:  Security
     Object Name:    D:
     Handle ID:      0x5bc
Process:
     Process ID:     0x1820
Permissions Change:
     Original Security Descriptor:
D:(A;OICI;FA;;;SY)(A;OICI;FA;;;BA)
     New Security Descriptor:
D:ARAI(A;OICIID;FA;;;SY)(A;OICIID;FA;;;BA)
(A;OICIID;FW;;;bob)
```

Figure 1: Example event showing the change in security permissions (ID 4670)

and services often contain specific debug messages that may help an expert understand what went wrong. There is no requirement to adhere to a certain standard and the information can be as brief or descriptive as the programmer decided. Fortunately, however, when creating event log entries related to security events, Microsoft have adopted and consistently adhered to an object-centric logging standard. This is where all objects related to the log entry (user, process, etc.) are clearly stated, which provide sufficient detail for the security analyst to understand the occurrence of this event. For example, if someone made ten failed login attempts into a server, the security event logs will contain ten events logs with $I = 4625$. Each entry will have the information about the account name, failure reason, date/time, source network address, port etc. Together with this data, an expert can determine if there was a security breach incident along with its kind and what security measures should be taken to avoid this in future.

Figure 1 provides an example event description for assigning new file system permissions. As the text details, the log ID is 4670, and the series of objects are listed below. This includes those related to the "Subject", "Object", "Process" and those related to the "Permissions Change". An example object is "Account Name" and the object value is "Bob".

## 4. Frequency Analysis

Performing a primitive frequency analysis of the event logs makes it possible to easily identify event types that are occurring both frequently and infrequently. However, although such primitive analysis is beneficial, the standard built-in event viewer does not provide such information.

The technique presented in this paper processes a series of events and counts the frequency occurrence of each type, denoted by their ID. The technique iterates over all available events and constructs a series of all unique event IDs and the frequency of their occurrence. More specifically, $C = \{c_1,c_2,\ldots,c_n\}$ where $c = \{u,f\}$ represents a unique event ID, $u$, and the frequency of occurrence, $f$. $c$ is ordered in ascending order based on frequency such that $f_1 \leq f_2$ from $c_1$ and $c_2$, respectively.

This allows us to identify events that are infrequent and manually determine if they are irregular or not. However, difficulty arises when deciding the cut-off threshold for frequencies that should be treated

as potentially problematic. Expert analysis would help separate the irregular from regular events, but in some cases such expert knowledge is not available. Therefore, in this paper a technique is presented which attempts to firstly classify frequency scores which are most likely to be anomalous or irregular. To perform this classification, Jenks natural breaks classification method [25] is used to determine the best arrangement of values into different classes. This is performed by minimising each class's standard deviation, whilst maximising the standard deviation between classes. The class with the minimum standard deviation (i.e lower frequency $f$ scores) is the class of events which have the lowest frequency of occurrence and are therefore to be treated as potentially irregular. To perform this the following classification function is used:

$$I(x)\colon \{1 \dots n\} \mapsto \{1 \dots k\} \tag{1}$$

where $n$ is the number of data samples, and $k$ is the number of classes where $k \leq n$. $S_j$ are the set of indices that map to class $j$. The minimal sum of the sums of standard deviations ($SDD_{n,k}$) is then calculated by:

$$SDD_{n,k} = \min_{I} \sum_{j=1}^{k} ssd(S_j) \tag{2}$$

The minimisation function is to select a classification (using $I$) such that the optimal number of classes are chosen resulting in a the minimal $SDD_{n,k}$, and thus the data is determined to be a 'natural fit'. The $ssd(S_j)$ is the sum of the squared deviations of the values of any index set $S$ calculated using the following equation where $A$ is an ordered set of $f$ scores.

$$ssd(S) = \sum_{i \in S} \left[ A[i] - \left( \frac{\sum_{i \in S} A[i]}{|S|} \right) \right]^2 \tag{3}$$

The minimisation function in Equation 2 is iteratively performed, with judgement being made as to whether introduce a new set of indices. The general rule here is to continue iterating and expanding the number of classes (and index positions) until the current $SDD_{n,k}$ is no longer a lower value that the previous $SDD_{n,k}$ value.

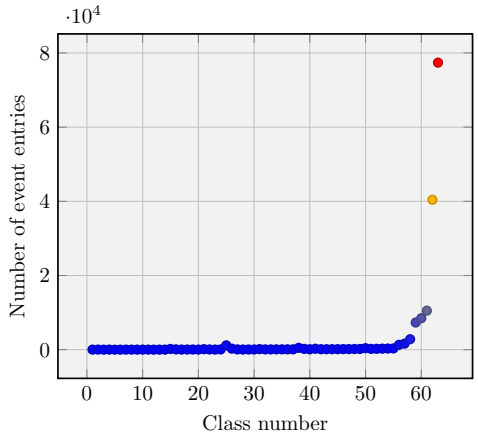


Figure 2: Number of event entries per class number

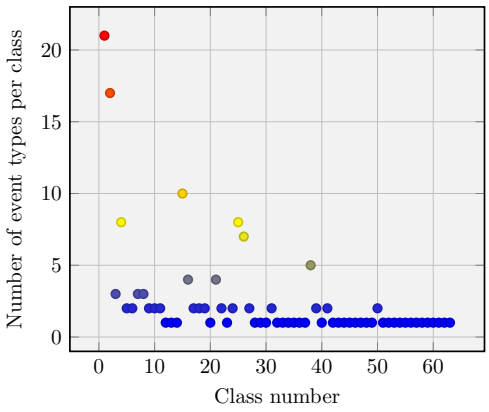


Figure 3: Number of event types per class

An illustrative example is now discussed whereby an event log is acquired containing 157,609 events in total. Performing frequency analysis on the event log determines that there are 181 unique event types that have been identified. Jenks analysis is then iteratively performed until convergence is reached where the minimum $ssd(S)$ value has been identified through increasing the number of classes ($j$). The technique converges at classifying the data into 63 classes. Figure 2 illustrates the number of total event entries within

each class, and Figure 3 illustrates the number of different event types within each class. The results are interesting as the number of event entries within the first class is low (at 1 per event type) and that the first class is made up of many different event types, whereas class 63 only contains one type of event. In the example provided, an example event from class one is 5973 which is reporting an issue with the Windows Store application, whereas the last class is made up of event type 4663 which denotes access to a file system resources.

This frequency analysis then requires interpretation by an expert to determine which class or classes contains events of interest. For example, classes containing a high frequency would be inspected to determine if there was some significant underlying problem causing the repeated logging of an event. Similarly, classes with a low frequency will be examined to identify any potential infrequent log entries which could denote suspicious information, and possibly even a security threat.

Although performing frequency analysis is useful for identifying event volumes, the provided example demonstrates how it alone is not sufficient to identify events that are reporting on irregular activity. Moreover, there is a chance that an infrequent event is reporting on irregular activity; however, it is also possible that a frequently occurring event is reporting on an irregular event. For this reason, the paper now focusses on identifying events that are irregular in terms of the objects they contain and their relationships with other events.

## 5. Test of Independence

The previous section has demonstrated a technique capable of classing event log entries based upon their frequency of occurrence, enabling the ability to provide enriched information to the user. However, there is a strong necessity to manually examine entries to verify those that are irregular and or anomalous. In this section, a technique is presented, which resolves this issue by autonomously identifying irregularities in the given event logs.

This section describes how irregularities in event log entries can be measured as the independence of a event's object, $o$, from an event type's, $t$, distribution of entries. This object-centred approach is taken due to interest in objects that are irregular for an event type. Furthermore, the occurrence of irregular objects provides a greater granularity of analysis. When programatically examining the event logs, each entry contains a series of string messages where key object-based information is present[1].

Using statistical analysis to determine irregular event entries creates the potential to categorise irregular, but correctly, event log entries. This is because in large multi-user systems there are many event logging mechanisms which may only generate few amount of event logs. However, identifying these less frequent events as irregular is still useful as it is important that they are monitored and reacted upon where necessary.

### *5.1. $\chi^2$ Analysis*

$\chi^2$ statistics are used to measure the lack of independence between $o$ and $t_j$ - which can then be compared to the $\chi^2$ distribution with one degree of freedom (as there are two groups) to judge extremeness [26]. The $\chi^2$ statistic is chosen as it is a well established technique for measuring independence. For example, it has been successfully used in text categorisation [27, 28], credit risk assessments [29], several medical studies regarding Ebola patients [30] and Dementia [31] etc. Other techniques are also available for measuring independence, such as *Paired t-test* [32] and *Pearson correlation* [33]. However, $\chi^2$ is not only computationally easy to compute, it is also a non-parametric test which makes no assumption regarding the distribution of the population [34]. This makes it a suitable candidate for the novel work presented in this paper. Using a two-way contingency table for object $o$ and event type $t_j$ where: $A$ is the number of times $o$ and $t_j$ co-occur, $B$ is the number of times $o$ occurs without $t_j$, $C$ is the number of times $t_j$ occurs without $o$, $D$ is the number of times neither $t_j$ or $o$ occur, $N$ is the total number of objects to examine.

[1]The `EventLogEntry` class within the Microsoft .NET framework contains a series of `ReplacementStrings`, which provide event details

| **Log Event ID** | **Number of objects** | $\chi^2_{avg}(e)$ | **Jenks class** |
|---|---|---|---|
| 153 | 6 | 1698 | 0 |
| 1073742727 | 2 | 2271 | 1 |
| 42 | 7 | 2512 | 2 |
| 32 | 4 | 2517 | 2 |
| 521 | 6 | 2725 | 3 |
| 25 | 4 | 3628 | 4 |
| 18 | 4 | 3628 | 4 |
| 37 | 9 | 4523 | 5 |
| 1073873154 | 2 | 4774 | 6 |
| 1073873155 | 2 | 4774 | 6 |
| 1073873153 | 2 | 4774 | 6 |
| 1073741861 | 3 | 5976 | 7 |
| 1073741856 | 3 | 5976 | 7 |
| 400 | 21 | 7248 | 8 |
| 403 | 21 | 7248 | 8 |
| 1026 | 50 | 7275 | 8 |
| 50037 | 4 | 8021 | 9 |
| 600 | 21 | 8488 | 10 |

Table 1: Example output from performing $\chi^2$ analysis a the Security event log of a personal computer, running Windows 10

From this a lack of independence measure between an object $o$ and event $t_j$ by:

$$\chi^2(o, t_j) = \frac{N(AD - CB)^2}{(A + B)(A + C)(B + D)(C + D)} \tag{4}$$

The $\chi^2$ statistic has a natural value of zero if $o$ and $t_j$ are independent. Therefore, it can be assumed that any object $o$ that occurs in event type $t_j$ with a $\chi^2$ value close to zero is either an anomaly or an irregular event entry. Following the calculation of $\chi^2$ scores, it is then useful to compute the mean $\chi^2$ for each object using the following equation where $l$ is the number of objects specified for an event, $e$:

$$\chi^2_{avg}(e) = \frac{1}{l} \sum_{j=1}^{l} \chi^2(o, t_j) \tag{5}$$

Similar to the desire to class frequency information in Section 4, there is a need to classify $\chi^2_{avg}(e)$ scores to help identify those that are irregular. Jenks natural breaks classification method is used once again to separate the one-dimension data in to classes. This enables information to be presented to the user in a way where minimal effort and time is needed to find potentially irregular event log entries.

The same event log as used for frequency analysis in Section 4 is now used to test the modelled and developed $\chi^2$ technique. The event log contains 157,609 and Table 1 provides an extract of the first 10 classes. Following this, the same Jenks analysis classification method is used to classify $\chi^2_{avg}(e)$ scores to identify the class with the lowest score and are most likely to be irregular. In the example presented in Table 1, the results are grouped in to 97 classes.

From Table 1 it is evident that those event IDs having the lowest $\chi^2_{avg}(e)$ scores are classified in the lower groups in the Jenks analysis. The event containing an ID of 153 informs that an IO operation issue has been detected during a writing operation, and potentially indicates a deteriorating hard drive health. The event with an ID of 1073742727 informs that a software protection service has stopped. Events within class number 10 (ID of 600) indicate that a service or a scheduled task has started under the authority of a different user. Clearly these events warrant further investigation by the administration and it would

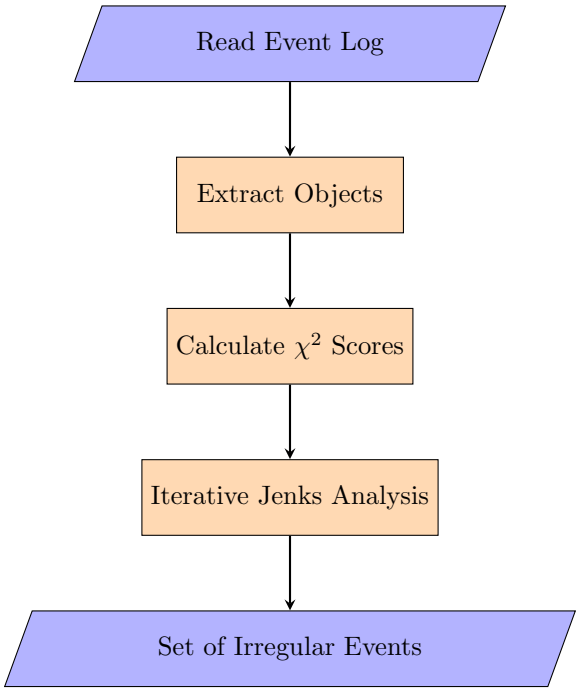


Figure 4: Process overview

appear that this mechanism has correctly identified events that warrant further investigation, without the administrator needing to perform an exhaustive manual search.

## 6. Implementation

The analysis technique has been implemented in Microsoft's C# language and utilises the .NET framework for extracting events from live systems, as well as processing them from a standardised stored Windows XML Event Log (EVTX) format. The software was executed on a machine with an Intel 3.6 GHz i7 CPU with 16GB of available RAM. Both code and datasets are made publicly available in the "Availability" section.

The implemented software includes code to perform $\chi^2$ and Jenks analysis, as presented in previous sections. Figure 4 provides an overview of the stages implemented in the software solutions. The first stage is where an event log is parsed and loaded into memory. This is then followed by the second stage where the internal representation is processed to extract the series of objects. This includes processing the event log description to extract key object information, excluding denoting text. For example, in security events the object information is often denoted by a descriptive phrase followed by the semi-colon (:). The use of a semi-colon creates potential to programmatically locate and extract object information.

Following this, an iterative algorithm is performed whereby $\chi^2$ scores are calculates for each object and event combination, and then combined into an average for each event. These $\chi^2$ scores for each event are then ordered in increasing order. An instance where an object and event are independent have a score of 0, and therefore greater $\chi^2$ scores indicate dependence.

The final process is to iteratively perform Jenks analysis, increasing the number of classes with each iteration. The purpose is to autonomously identify the best classification and to identify whether class 0 –the one containing the lowest $\chi^2$ scores– contains a series of anomalies. The output from the software is a set of irregularities from class 0.

## 7. Experimental Analysis

In this section, the ability of the presented technique is examined and discussed in terms of its ability to maintain good performance with increasing log sizes, as well as its ability correctly identify irregular log entries of interest. This section includes the three following analysis sections:

- Scalability analysis where computation time is presented for event logs with an increasing number of irregularities, as well as establishing at what point the number of irregular events results in them no long being identifiable as irregular. I.e., convergence has been identified.
- Analysis on previously 20 live, unseen event logs. No ground truth knowledge is available for the event logs and they are processed to identify potential irregularities. This section discusses the identified irregularities and suggests why they contain information that is of security relevance.
- As identified in the following sections, a large number of the 20 event logs do not contain any identifiable irregular event entries. The third section includes introducing synthetic irregular event entries into each of the 20 event logs.

### *7.1. Scalability*

The main concern regarding scalability of the technique is around at what point an event log entry, if frequently reoccurring, is no longer identifiable (i.e., a state of convergence has been reached). To demonstrate the suitability of $\chi^2$ analysis for identifying irregularities on an albeit trivial system, an example is provided where an event log is constructed to contain synthetic irregular events. The events all report the allocation of new file system permissions, involving 10 unique users across 10 unique directories. Therefore the distribution of permissions is that each user will be allocated permissions on 10 directories. An irregular permission is then added where a new user (herein referred to as the test user) is allocated permission on one of the 10 directories. The number of irregular permissions is then increased until 10, and at this point the user assigned irregular permissions should be consistent with the 10 regular allocations. The experimental event logs are created using a simple Powershell script to create users, directories, and assign permissions.

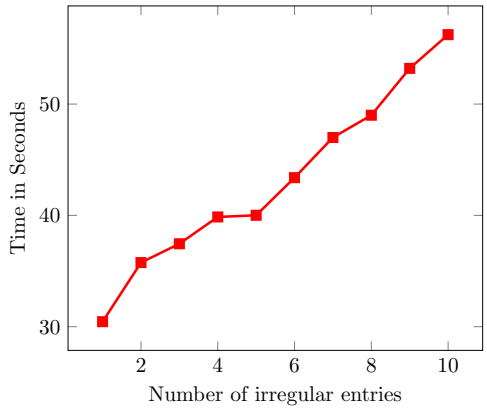


Figure 5: Required computation time for computing $\chi^2$ scores.

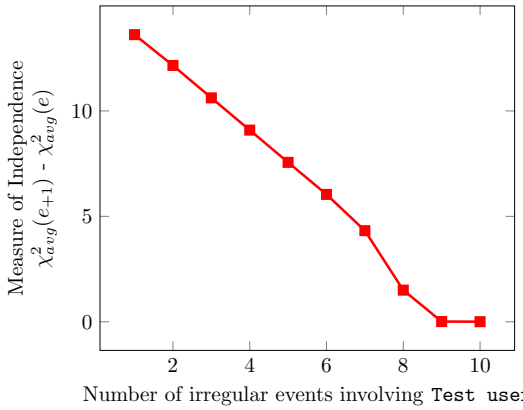


Figure 6: Results from calculating the independence of test user with an increasing number of associated permissions. The graph shows the difference between $\chi^2_{avg}(e)$ and the next most significant event entry. Therefore, the higher the number the better.

Figure 5 illustrates the computation time required for each event log. The phrase 'irregular' events refers to the number of deliberately assigned permissions that are used to simulate infrequent event entries. It is noticeable in the figure that computation time is increasing as the number of irregular event entries increases. Calculation of $\chi^2$ scores is linear and has a complexity of $O(n)$, where $n$ is the number of events. For example,

an event log dataset with 10 users, 10 directories, and each user granted access to each directory, contains a total of 850 event entries made up of 11,130 objects. It took thirty seconds on a 3.6 GHz i7 CPU with 16GB of available RAM to process the results. Note that the number of events is considerably higher than 100 (1 for each user on each directory) as other events are also recorded during the process. For example, events also report on requesting the access control list from an object.

Figure 6 illustrates the results from this experiment. The graph shows the difference between $\chi^2_{avg}(e)$ and the next most significant event entry. The x-axis in the graph represents the total number of irregular event log entries. However, as the test user is not involved with any other event log entries (i.e., creating user accounts), it can also be stated that these are the total number of irregular log entries for the test users. This allows us to establish by what margin $e_j$ is independent from $e_{j+1}$. From the graph it is evident that the measure of independence is decreasing as the number of event entries involving the test user has increases. The experiment was stopped at 10 event entries as the total number of event entries reached 1020. Even though the analysis is still suggesting the test user has the highest measure of independence, it is no longer feasible to consider entries involving the test user as irregular. The reason that test user still has a high measure of independence is because the remaining 416 log entries are distributed across 10 users and directories. The results presented in Figure 6 also demonstrate the potential implication of incorrect classification of regular event entries should an event occur less frequently than any of the others within the system. For example, from Figure 6 it can be seen that an event is more likely to be classified as an anomaly if it appears infrequently when compared to other event entries within the system. It can therefore be deduced that an infrequent yet correct event log entry is likely to be incorrectly classified if there are not enough event log entries containing the same objects.

### *7.2. Live System Analysis*

In this section, experimental analysis is performed on the event logs acquired from 20 number of computers. Each computer is operating in a live environment and the event logs have been acquired with owner permission. Dataset 1-12 and 14-15 are acquired from machines with the Windows 7 operating system (OS) and work in a multi-user environment. The machines are typically used by university students. The students belong to either under- or post-graduate programs and have different access rights and restrictions. Typically, post-graduate students have a higher level of permission to develop, configure, and install software. The machines are mostly used for document writing, Internet surfing, and the development of software applications. Datasets 13, 16-20 are taken from Windows 10 based machines. They have the similar kinds of usage as of Windows 7 machines, but the frequency of different users is higher. These event logs data is gathered from 5 different university laboratories.

#### *7.2.1. Results*

Table 2 describes the characteristic of each event log. The table demonstrates the number of events within each dataset and the total number of objects. In addition, the table also presents the number of unique event types and objects as the developed technique takes steps to minimise processing time and memory consumption. From analysing the table, it is evident that this experimental analysis is based on a diverse range of event logs, ranging from those with around 300 events containing more than 5k objects to those with other 25k events with more than 280k objects. As expected, the processing time ranges considerably from 2 to almost 1,200 seconds. Interestingly, it is not necessarily the event logs with a large number of events and objects that are the most time consuming. For example, the largest dataset (19) with 26k events and 282k objects is processed in 454 seconds, whereas dataset 18 of less than half the size (8.8k events, 1331k objects) takes almost twice as long to analyse 950 seconds. The reason behind this difference is due to the number of Jenks iterations performed before arriving at the point of convergence, where it is not advantageous to further divide the data into more classes. In this specific example, dataset 19 took 11 iterations, whereas 18 required 75. It is also worth noting that the number of unique objects for dataset 18 and 19 is 1.3k and 1.7k, respectively. Considering that the technique only processes unique objects for efficiency, it can be seen that the technique scales well to datasets of increasing sizes. The dataset requiring the longest amount of time is dataset 3, containing 11.9k events and 13.6k unique objects. This is the highest combination seen in all 20 datasets and required 1.2k seconds to process.

| **PC / Dataset Number** | **Number of events** | **Number of objects** | **Number of unique event types** | **Number of unique objects** | **CPU time (s)** | **Number of Classes** | **Number of events in class 0** | **Contains Irregular Events** |
|---|---|---|---|---|---|---|---|---|
| **1** | 7752 | 136384 | 29 | 10238 | 590 | 26 | 7247 | No |
| **2** | 8249 | 142617 | 29 | 9786 | 700 | 34 | 569 | No |
| **3** | 11953 | 212363 | 26 | 13602 | 1217 | 26 | 5002 | No |
| **4** | 7725 | 135976 | 28 | 10230 | 462 | 12 | 7546 | No |
| **5** | 7706 | 135739 | 28 | 9206 | 467 | 21 | 7182 | No |
| **6** | 7884 | 138820 | 28 | 10320 | 589 | 24 | 7326 | No |
| **7** | 8258 | 145702 | 28 | 10714 | 725 | 32 | 2508 | No |
| **8** | 9718 | 172028 | 29 | 12009 | 914 | 30 | 6091 | No |
| **9** | 10203 | 180791 | 28 | 12353 | 778 | 17 | 9834 | No |
| **10** | 7781 | 137032 | 28 | 11227 | 564 | 24 | 7249 | No |
| **11** | 9858 | 174517 | 29 | 12544 | 846 | 19 | 9488 | No |
| **12** | 9767 | 172953 | 27 | 11667 | 863 | 27 | 6222 | No |
| **13** | 13022 | 173704 | 28 | 7729 | 751 | 24 | 2142 | No |
| **14** | 293 | 5252 | 5 | 712 | 2 | 44 | 3 | *Yes* |
| **15** | 298 | 5342 | 5 | 712 | 4 | 111 | 1 | *Yes* |
| **16** | 568 | 7899 | 4 | 727 | 5 | 61 | 1 | *Yes* |
| **17** | 10948 | 163103 | 15 | 7068 | 464 | 11 | 5581 | No |
| **18** | 8817 | 113183 | 20 | 1321 | 958 | 74 | 72 | No |
| **19** | 26001 | 282581 | 16 | 1753 | 456 | 11 | 9020 | No |
| **20** | 386 | 6902 | 6 | 1131 | 9 | 161 | 1 | *Yes* |

Table 2: Example output from performing $\chi^2$ analysis

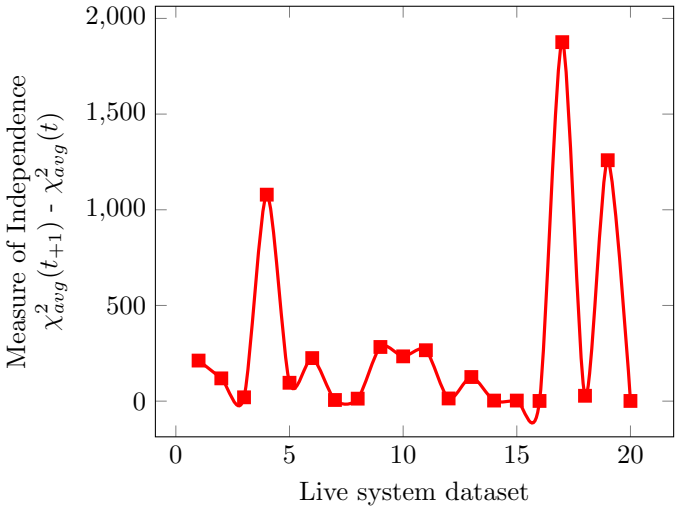


Figure 7: Results from performing $\chi^2$ on the 20 event logs

Figure 7 presents the difference between $\chi^2_{avg}(e_j)$ and $\chi^2_{avg}(e_{j+1})$. More specifically, the difference between the average score for the event entries identified in the first class – those identified as irregular – and those in the next class with the closest average score. This is useful as it quantifies by how much they are identified as irregular. The figure details that the majority of the 20 datasets have a difference score of less than 400. Interestingly, 4, 17 and 19 all have difference scores of greater than 1,000. After cross-referencing this information with the data provided in Table 2, it is clear that this is potentially due to the iterative implementation of the Jenks natural breaks algorithm has converged with a low number of classes when compared to all other datasets. More specifically, 17, 11 and 11 for dataset 4, 17 and 19, respectively.

Both Table 2 and Figure 7 demonstrate the characteristics and performance measures of the 20 datasets. However, it is now necessary to consider the technique's ability to identify irregular events of interest. As we have no ground truth knowledge regarding contents of each dataset, determining overall accuracy without performing an exhaustive audit of each dataset is challenging. Due to the restriction on human expert knowledge, performing such a manual audit is not feasible. Furthermore, the technique is an unsupervised approach and is expected to identify irregular events of potential interest without any prior expert guidance. In the following section, empirical analysis is performed on the events identified within each dataset. The analysis will interrogate each event individually to determine whether it is irregular and of potential interest. From this analysis it is possible to calculate a measure of accuracy, i.e., the fraction of events identified to be irregular that are both correctly and incorrectly identified.

After analysing the results from processing the 20 event logs, it is evident that no irregular events have been identified within a significant portion of the event logs. More specifically, 15 of the event logs have been found to not have any irregular events within them. A manual exploration has taken place to confirm that this is that case, and in all cases this outcome has been identified as valid. In the remainder of this section, a detailed analysis of the outcomes for event logs 14, 15, 16, and 20 are presented.

The irregular events identified within event logs 14 and 20 are all of event type 5447, which denotes that a "Windows Filtering Platform filter" has been changed. Event logs 14 and 20 have 3 and 1 occurrences identified as irregular. Interestingly, these events have been identified not because of the frequency, but because the objects that are reported in the events occurring infrequently with this event ID. Events of type 5447 occur 576 and 762 times in total for dataset 14 and 20, respectively. The 5447 events within event logs 14 and 20 have $\chi^2_{avg}(e)$ of less than 3. In both event logs, the occurrence of objects such as "TermServiceLOM", "ALE Receive/Accept v6 Layer", and "NT AUTHORITY\LOCAL SERVICE" are identified as infrequent and rarely occurring with events of type 5447. It is therefore possible to establish that these objects fail a test of dependence, and thus are classified as irregular.

Event log 15 contains one event of type 4726 that denotes that a user account was deleted. This event was identified as been irregular as although many other account deletion events are present within the log, the one identified as irregular contains the use of a "TUser0". In a similar fashion, "TUser0" was involved in many other events in the event log, but is not linked to any other account deletion events. Although it is not possible to state whether this event is of any significance to an investigator, it does demonstrate the techniques ability to adequately identify irregular event entries. More specifically, although the technique is good at finding poorly connected object and event types, it is not possible to state whether it contains any significant security information. However, based on empirical observations performed in this paper, it can be stated that the security events identified do contain information pertinent to increasing a system's security.

One irregularity has been identified in event log 16. This is event 4798 and reports that a user's local group membership was enumerated. This event is automatically generated by the system once a process enumerates a user's security-enables local groups. More specifically, the process is analysing the security group associations of a user. In the first instance it would appear that this event is not out of the ordinary as processes will frequently enumerate a user's group associations during requesting permission to resources and services. However, this particular event is irregular due to its associated objects. This event contains the "Administrator" group which is not irregular; however, what is irregular is that it appears within the same event as the workstation "CWG05-04". This is the first time this combination occurs and hence it being identified as irregular.

This empirical analysis has demonstrated that irregular event log entries can be detected that are irregular in terms of their object relationship. This is significant as simply considering event frequency would not identify these events. In the presented examples, all the identified event types are frequently occurring. Although this analysis has only identified irregular events within four out of sixteen of the event logs, it is still beneficial to be able to identify that there are no irregular events within the logs.

### *7.3. Live System and Synthetic Irregular Events*

In the previous section, 20 event logs acquires from live systems have been analysed. A key finding is that the technique has not identified any irregular events in 16 of the logs. Although, manual analysis has taken place to ensure that this is correct, it is not fully known if any irregular security events have been missed. More specifically, in the analysis, classes containing the lowest $\chi^2_{avg}(e)$ values are manually inspected to determine if any irregular security events have been identified in classes of a low number (i.e., those greater than 0). However, a key limitation of this analysis is that irregular events may be present in classes containing a higher $\chi^2_{avg}(e)$ score, and due to the time required to perform a full manual audit, irregular security events may have gone undiscovered. For this reason, this section introduces analysis whereby synthetic irregular security events are added and used as 'ground-truth' knowledge to determine the techniques accuracy.

In performing this synthetic analysis, the following methodology is followed:

1. First, the previous 20 event logs are processed in turn to identify what security event types are present. This involves matching against a known list of security event IDs [2]. It is important to identify what security events are available within the system, and to use them to create synthetic and irregular event logs. Selecting an event type that does not exist within the log would yield poor results as the technique is designed to identify irregularities in the relationship between objects and events, and not the occurrence of a unique security event.
2. Identify the frequency of each security event type, and select the 10 most frequent. A maximum of 10 is used due to the findings in Figure 6, whereby after 10 irregular events the difference between the average values of class 0 and 1 converge. The most frequent are chosen as we want to identify the technique's ability to identify irregular events log entries, where an irregular event is defined by its unusual set of objects. In the instance where 10 different event types are not available, we use the number available, providing the occurrence of the event type is greater than 10.

[2] A list of all security event IDs was identified at the following URL:`https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/`

| PC / Dataset Number | Number of Irregularities | tp | fp | tn | fn | tpr | fpr | tnr | fnr | accuracy |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 7 | 7 | 0 | 7752 | 0 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 |
| 2 | 9 | 9 | 0 | 8249 | 0 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 |
| 3 | 6 | 2 | 2664 | 9289 | 4 | 0.33 | 0.22 | 0.78 | 0.67 | 0.56 |
| 4 | 6 | 6 | 0 | 7725 | 0 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 |
| 5 | 6 | 6 | 2706 | 5000 | 0 | 1.00 | 0.35 | 0.65 | 0.00 | 0.82 |
| 6 | 7 | 7 | 0 | 7884 | 0 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 |
| 7 | 6 | 6 | 0 | 8258 | 0 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 |
| 8 | 6 | 6 | 226 | 9492 | 0 | 1.00 | 0.02 | 0.98 | 0.00 | 0.99 |
| 9 | 6 | 6 | 0 | 10203 | 0 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 |
| 10 | 6 | 6 | 0 | 7781 | 0 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 |
| 11 | 6 | 5 | 0 | 9858 | 1 | 0.83 | 0.00 | 1.00 | 0.17 | 0.92 |
| 12 | 7 | 7 | 419 | 9348 | 0 | 1.00 | 0.04 | 0.96 | 0.00 | 0.98 |
| 13 | 10 | 2 | 1632 | 11390 | 8 | 0.20 | 0.13 | 0.87 | 0.80 | 0.54 |
| 14 | 1 | 1 | 288 | 5 | 0 | 1.00 | 0.98 | 0.02 | 0.00 | 0.51 |
| 15 | 1 | 1 | 293 | 5 | 0 | 1.00 | 0.98 | 0.02 | 0.00 | 0.51 |
| 16 | 1 | 1 | 0 | 586 | 1 | 0.50 | 0.00 | 1.00 | 0.50 | 0.75 |
| 17 | 6 | 2 | 0 | 10948 | 4 | 0.33 | 0.00 | 1.00 | 0.67 | 0.67 |
| 18 | 10 | 10 | 77 | 8740 | 0 | 1.00 | 0.01 | 0.99 | 0.00 | 1.00 |
| 19 | 10 | 10 | 8 | 25993 | 0 | 1.00 | 0.00 | 1.00 | 0.00 | 1.00 |
| 20 | 1 | 1 | 132 | 254 | 0 | 1.00 | 0.34 | 0.66 | 0.00 | 0.83 |
| | | | | | **Average** | **0.86** | **0.15** | **0.85** | **0.14** | **0.85** |

Table 3: Results from performing experimental analysis

3. Synthetic event entries are then inserted using a synthetic set of objects, which are unique to the event's occurrence in respect to the set of all objects related to events of this specific type. For the purposes of this analysis, a `test` object is introduced and assigned to the synthetic event.
4. Perform analysis on each of the 20 event logs, using ground-truth knowledge of the synthetic irregular events to identify: True Positive Rate (*tpr*): the fraction of irregular events correctly identified as being irregular; False Positive Rate (*fpr = 1 - tnr*): the fraction of regular event logs incorrectly identified as being irregular; True Negative Rate (*tnr*): the fraction of regular event logs entries correctly identified as regular; False Negative Rate (*fnr = 1- tpr*): the fraction of irregular event log entries incorrectly classified as regular; and finally, *Accuracy* is reported as the fraction of all samples correctly identified. More specifically, $Accuracy = \frac{tpr+tnr}{tpr+tnr+fpr+fnr}$

Table 3 presents the results from performing the analysis. As evident in the table, the number of irregularities is in most cases (apart from dataset 13, 18, and 19) is less than 10. This indicates that that in the majority of the event log datasets, there are fewer than 10 security event types. The lowest occurring in dataset 20 whereby there is only 1 security event type identified.

It is evident in Table 3 that the *tpr* is high due to a large potion of irregularities being correctly identified. Those datasets that have a poor *tpr* –specifically 3, 13, and 17 and all have a *tpr* below 0.40 – are those identified to have a smaller difference in the measure of independence between class 0 and 1 ($\chi^2_{avg}(t_1)$ - $\chi^2_{avg}(t_0)$) in Figure 7. This inability to detect irregular events due to the numeric insignificant of their $\chi^2_{avg}(e)$ score. More specifically, there is a small, and sometime no, difference between all the events identified to be part of class 0 and subsequent classes (1, 2, etc.).

This above reason is why the technique sometimes has a high *fpr*. There are, however, also datasets with high a *fpr* that have a good *tpr*. For example, dataset 14 and 15 have a *tpr* of 1 but have a high *fpr* due to many regular events been incorrectly identified. This incorrect classification is due to the analysis

technique not identifying any statistical difference between regular and irregular event log entries. From cross referencing these results with those presented in Table 2, it is clear that datasets with a high *fpr* have both a substantially lower number of event and unique event types.

The *tnr* and *fnr* are also good apart from those event logs of a small size or those that have poor *tpr* and *fpr* values. The overall accuracy reports the fraction of events which are correctly identified (either as correct or incorrect). The average accuracy over all datasets is 85%; however, it should be noted that datesets significantly impacting the overall accuracy are those with few event entries and unique event types.

## 8. Conclusion

This paper presents a novel technique and its implementation in software for automatically identifying irregular event logs entries, which could indicate suspicious and malicious activity in a system. Currently, the implementation is done for Microsoft based operating systems using C# language and .NET platform. The proposed solution requires the input in a structured manner to perform accurate analysis. Each event log is modelled into an objects-based entity. Each entity contains a group of timestamp, event ID and set of relevant objects such as account names, processes etc. In this paper, first the frequency of each unique event log is determined and further arranged into different classes using Jenks natural breaks method. The problem with this approach is that it requires expert knowledge and manual work to verify whether low frequency events are irregular and of significance.

In order to produce a more sophisticated and useful solution, statistical analysis is performed using the well-established $\chi^2$ test along with Jenks method to categorise the one-dimensional data to identify groups of entries with low scored. The $\chi^2$ statistic has a natural value of zero, given the objects are fully independent. Therefore, if the mean $\chi^2$ value of event log (determined by its objects) is closer to zero, it implies the event log is relativity independent from others, and hence classed as irregular. The proposed solution is tested by acquiring event logs from a multi-user network of machines in a university environment. The results are promising as the solution was able to identify irregular event logs based on object relationship in a reasonable amount of time. For example the largest event log processes with in excess of 210k event logs requires over 1200 seconds. The proposed tool is beneficial as the manual process of identifying irregular event requires expert knowledge, which might not be present or affordable. Further experimentation has demonstrated the technique's ability to identify synthetic irregularities with an average 85%. The produced technique can provide both experts and non-experts alike with a technique capable of identifying potential events of interest much more quickly than performing a manual audit.

## 9. Availability

All source code and event logs used in this paper are available at: `bttp:\selene.hud.ac.uk\scomsp2\event`

## 10. References


[1] S. Khan, S. Parkinson, Causal connections mining within security event logs, in: Proceedings of the 9th International Conference on Knowledge Capture, ACM, 2017.
URL `http://eprints.hud.ac.uk/id/eprint/33841/`

[2] S. Parkinson, A. Crampton, Identification of irregularities and allocation suggestion of relative file system permissions, Journal of Information Security and Applications 30 (2016) 27–39.

[3] D. Schauland, D. Jacobs, Managing the windows event log, in: Troubleshooting Windows Server with PowerShell, Springer, 2016, pp. 17–33.

[4] D. C. Luckham, B. Frasca, Complex event processing in distributed systems, Computer Systems Laboratory Technical Report CSL-TR-98-754. Stanford University, Stanford 28.

[5] W. Van der Aalst, T. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE Transactions on Knowledge and Data Engineering 16 (9) (2004) 1128–1142.

[6] W. M. Van Der Aalst, H. A. Reijers, M. Song, Discovering social networks from event logs, Computer Supported Cooperative Work (CSCW) 14 (6) (2005) 549–593.

[7] M. Aharon, G. Barash, I. Cohen, E. Mordechai, One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs, Machine Learning and Knowledge Discovery in Databases (2009) 227–243.
[8] C. C. Aggarwal, J. Han, Frequent pattern mining, Springer, 2014.
[9] R. Vaarandi, A breadth-first algorithm for mining frequent patterns from event logs, Intelligence in Communication Systems (2004) 293–308.
[10] C. Borgelt, An implementation of the fp-growth algorithm, in: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations, ACM, 2005, pp. 1–5.
[11] H. Li, Y. Wang, D. Zhang, M. Zhang, E. Y. Chang, Pfp: parallel fp-growth for query recommendation, in: Proceedings of the 2008 ACM conference on Recommender systems, ACM, 2008, pp. 107–114.
[12] L. Zhou, Z. Zhong, J. Chang, J. Li, J. Z. Huang, S. Feng, Balanced parallel fp-growth with mapreduce, in: Information Computing and Telecommunications (YC-ICT), 2010 IEEE Youth Conference on, IEEE, 2010, pp. 243–246.
[13] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.-C. Hsu, Freespan: frequent pattern-projected sequential pattern mining, in: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2000, pp. 355–359.
[14] X. Yan, J. Han, gspan: Graph-based substructure pattern mining, in: Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on, IEEE, 2002, pp. 721–724.
[15] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: ACM sigmod record, Vol. 29, ACM, 2000, pp. 1–12.
[16] C. K.-S. Leung, Q. I. Khan, Z. Li, T. Hoque, Cantree: a canonical-order tree for incremental frequent-pattern mining, Knowledge and Information Systems 11 (3) (2007) 287–311.
[17] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.
[18] A. J. Weijters, W. M. Van der Aalst, Rediscovering workflow models from event-based data using little thumb, Integrated Computer-Aided Engineering 10 (2) (2003) 151–162.
[19] W. Xu, L. Huang, A. Fox, D. Patterson, M. I. Jordan, Detecting large-scale system problems by mining console logs, in: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, ACM, 2009, pp. 117–132.
[20] P. Barham, R. Isaacs, R. Mortier, D. Narayanan, Magpie: Online modelling and performance-aware systems., in: HotOS, 2003, pp. 85–90.
[21] D. Ayers, A second generation computer forensic analysis system, digital investigation 6 (2009) S34–S42.
[22] A. R. Arasteh, M. Debbabi, A. Sakha, M. Saleh, Analyzing multiple logs for forensic evidence, digital investigation 4 (2007) 82–91.
[23] S. Peisert, S. Karin, M. Bishop, K. Marzullo, Principles-driven forensic analysis, in: Proceedings of the 2005 workshop on New security paradigms, ACM, 2005, pp. 85–93.
[24] H. S. Njemanze, P. S. Kothari, Real time monitoring and analysis of events from multiple network security devices, uS Patent 7,376,969 (May 20 2008).
[25] G. F. Jenks, The data model concept in statistical mapping, International yearbook of cartography 7 (1) (1967) 186–190.
[26] P. E. Greenwood, A guide to chi-squared testing, Vol. 280, John Wiley & Sons, 1996.
[27] Y. Yang, J. O. Pedersen, A comparative study on feature selection in text categorization, in: ICML, Vol. 97, 1997, pp. 412–420.
[28] K. Aas, L. Eikvil, Text categorisation: A survey, Raport NR 941.
[29] G. V. Attigeri, M. Pai, R. M. Pai, Credit risk assessment using machine learning algorithms, Advanced Science Letters 23 (4) (2017) 3649–3653.
[30] A. Colubri, T. Silver, T. Fradet, K. Retzepi, B. Fry, P. Sabeti, Transforming clinical data into actionable prognosis models: machine-learning framework and field-deployable app to predict outcome of ebola patients, PLoS neglected tropical diseases 10 (3) (2016) e0004549.
[31] C. D. Smyser, N. U. Dosenbach, T. A. Smyser, A. Z. Snyder, C. E. Rogers, T. E. Inder, B. L. Schlaggar, J. J. Neil, Prediction of brain maturity in infants using machine-learning algorithms, NeuroImage 136 (2016) 1–9.
[32] H. Hsu, P. A. Lachenbruch, Paired t test, Wiley Encyclopedia of Clinical Trials.
[33] J. Benesty, J. Chen, Y. Huang, I. Cohen, Pearson correlation coefficient, in: Noise reduction in speech processing, Springer, 2009, pp. 1–4.
[34] N. Balakrishnan, V. Voinov, M. S. Nikulin, Chi-squared goodness of fit tests with applications, Academic Press, 2013.