Enhanced Event Time-Lining for Digital Forensic Systems

Colin Symon

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
Computer Networks & Distributed Systems (Hons)

School of Computing

November 2009

Authorship Declaration

I, Colin Symon, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:

Date: 23rd November 2009

Matriculation no: 05002526

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

In a digital forensics investigation, log files can be used as a form of evidence by reconstructing timelines of the computer system events recorded in log files. Log files can come from a variety of sources, each of which may make use of proprietary log file formats (Pasquinucci, 2007). In addition, the large volume of information to be filtered through can make the job of forensic examination a difficult and time consuming task.

The aim of this thesis is to explore methods of logging and displaying event information which is gathered from computer systems, specifically in relation to the collection, correlation and presentation of log information. By means of a literature review, it has been found that by correlating and storing log information in a central log database it should be possible to construct a system which can access this information and present it in the form of a timeline to the investigator. The important contribution that visualisation techniques can bring to log analysis applications has been made by Marty (2008, p.5) by stating that “a picture is worth a thousand log records”.

A prototype system has been produced which makes use of the latest technologies to enhance current methods of displaying log data, such as those employed by the Microsoft Windows Event Viewer. The prototype system, developed using a rapid prototyping methodology, separates the log management process into collection, correlation and storage, and presentation. Through use of a standard XML log format and central storage of log information in a Microsoft SQL Server 2008 database, the prototype aims to overcome the issue of proprietary log formats and the difficulty in correlating data obtained from different sources. A log and timeline viewer application has been developed using C#, Windows Presentation Foundation and .NET Framework technologies, enabling the digital forensics investigator to filter event records and visualise timelines of events by means of bar, line and scatter charts.

Through the means of user evaluation it has been found that the prototype system improves upon the Microsoft Windows Event Viewer from overview and filtering perspectives. By means of technical experimentation, it has been found that there are scalability issues with the way in which the prototype system imports log information contained within XML files, into the database component. The time taken to import log records, of various sizes, into the database was measured. It was found that for files larger than 2MB, the time taken was longer than two users, of the seven who gave feedback on of the system, would be prepared to wait. Further development into the visualisation of timelines has been suggested as the prototype system is somewhat limited in its ability to provide details of the links between digital events.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Context

Digital forensics involves the analysis and interpretation of data gathered from computer systems which are being investigated. Typically, this involves analysing a variety of data sources such as files, log data, web browser histories and other information stored by the computer's operating systems and the applications run on it. Some of this information, such as debug logs, is intended to provide debugging information whereas other logs are generated solely with the purpose of recording information which may be of use in the event of a digital investigation (Casey, 2008) – this would typically be done in a secure corporate environment where a degree of monitoring is accepted.

Traditionally, the process of gathering this digital evidence has been time-consuming and without automated techniques investigators have difficulties establishing links between events, and filtering out data of interest. Even with systems such as the Microsoft Windows Event Viewer, there is a lack of visualisation technology which can make identifying patterns and anomalies difficult unless the intrusion time line is already known.

Digital Forensic Software, such as EnCase (Guidance Software, 2008), provide a means for the digital investigator to gather potential evidence in an automated and forensically sound manner. By taking this system further, and with the use of visualisation technologies such as bar charts, it should be possible to enhance current systems and provide investigators, as well as other interested parties, with the facilities required to establish timelines of events and identify unusual activity without resorting to manually reading through large log files, browser histories and other digital artifacts.

1.2 Aim and Objectives

The aim of this report is to explore methods of logging and displaying event information which is gathered from computer systems, specifically in relation to the collection, correlation and presentation of log information. In order to support this aim and provide a means of critical evaluation, the following objectives were specified:

- Review current literature and technology relating to digital forensics, log management and visualisation of log data. Determine the current issues and suggested solutions in relation to event recording, storage and presentation.

- Design a prototype system which enhances current methods of resolving difficulties surrounding the collection, correlation, storage and presentation of log data by implementing features suggested by relevant literature.
- Produce the prototype as per the design and test the system's reliability, impact on the host system and performance.
- Seek user feedback to uncover patterns of opinion and further test the prototype by assessing users' experiences.

1.3 Structure

This thesis has the following structure:

Chapter 1 – Introduction provides background information, puts the work in context and details the aim and objectives of the project.

Chapter 2 – Literature review investigates and discusses the work of others being carried out with relation to digital forensics – in particular the digital investigation process, log management and visualisation of log data.

Chapter 3 – Design of the prototype and justification for design decisions. Explanation of the tests to be carried out, any expected results and the reasons for performing the tests.

Chapter 4 – Implementation of the prototype. Discussion of how the implementation progressed, problems encountered and any changes made to the design based on experiences.

Chapter 5 – Details of how the testing was carried out and the results found.

Chapter 6 – Evaluation of the prototype's design post-implementation and discussion of the test results in relation to the usability, performance and robustness of the solution.

Chapter 7 – Conclusions and Future Work, reviewing the main findings and results and discussion of where further work could be carried out.

2 Literature/Technology Review

2.1 Introduction

The purpose of this literature review is to gain an understanding of Digital Forensics in general in order to establish the context in which event time-lining systems are used. The review then focuses on how data can be gathered and used in an investigation through the use of event logging and management tools and then be presented using time-lining visualisation techniques.

By reviewing how technologies have been developed and issues which have been identified, it will be possible to design a system which, using the latest standards and technologies, meets the goal of improving upon the ability of current systems to gather, correlate and present event information in the form of a timeline.

2.2 Background to Digital Forensics

Many computer security tools have been created which aim to prevent criminals gaining entry to systems or limiting the damage they can do. Products such as Firewalls, Anti-Virus and Web-Filtering aim to prevent damage to the systems in the first place and help to enforce an organisation’s Information Security Policies. Even where these security products have not prevented an intrusion, the event records they hold are often the source of evidence which can form part of a digital forensics investigation.

There are various definitions of digital forensics in literature. US-CERT provides a definition in their paper on Computer Forensics:

> “Forensics is the process of using scientific knowledge for collecting, analyzing, and presenting evidence to the courts” (US-CERT, 2008)

This definition of forensics covers the essence of digital forensics – to investigate the incident, by extracting digital evidence, and present the findings in a way that is acceptable in a court of law or in a company’s internal disciplinary hearing.

Just as a police detective pieces together events in order to build up a picture of a robbery and the events leading up the act, a digital forensic investigator analyses a variety of pieces of digital evidence and pieces them together to form a timeline. In digital forensics, evidence might include logs of file access, operating system events, and firewall activities, copies of emails and records of web access. According to the Home Office Voluntary Code of Practice (2003), recording of digital events forms part of legislation, such as the Regulation of Investigatory Powers Act 2000 and the Anti-Terrorism, Crime & Security Act 2001. The issue of retaining increasing amounts of information makes the job of the digital investigator ever more challenging as there can be millions of pieces of data to extract and filter through in order to gather relevant evidence. To facilitate the collection and filtering process, digital forensic investigators typically use software tools which can gather evidence from a number of sources and present it in a consistent format, such as a timeline.

2.3 Investigation Process

Forensic investigations, in general, consist of five main steps (Marcella & Menendez, 2008, p. 5):

1. Identification - The forensic investigator needs to be able to identify potential sources of evidence. The Electronic Crime Scene Investigation Guide: A Guide for First Responders (National Institute of Justice, 2001, pp. 10-11) suggests a number of potential sources of evidence held on computer systems; user-created files such as documents and emails, user-protected files such as compressed and encrypted files and computer-created files such as a web browser’s cookies, history files and log files.

2. Collection - At this stage the investigator gathers together the evidence from the various sources identified at the previous step.

3. Preservation - Copies of digital evidence are made. Marcella and Menendez (2008, p.6) suggest that this step is carried out to ensure the evidence is maintained in its original form, to enable the investigator to carry out their work without damaging the evidence and in the event that something goes wrong the investigator can return to the original state by making a new copy of the original evidence.

4. Interpretation – Processing the collected digital artifacts by:

   > “Determining its integrity, feasibility, usefulness to provide an opinion on the relevance of the electronic evidence to the case at hand” (Marcella & Menendez, 2008, p. 6).

5. Communication - This is the final stage of the investigation process. In the case of a legal inquiry, the investigator may have to explain their findings in a court of law.

2.4 Log files as a source of evidence

Use of log files enables a digital forensic investigator to trace events. In order to do this “we need at least to record Who has done What and When” (Pasquinucci, 2007). It is suggested that “log data is an abundant, freely available resource that is not current utilised or exploited” (Porter, 2003). Porter (2003) highlights the problem of identifying unusual activity within audit trails by stating that the average delay in discovering internal fraud is 18 months. Porter (2003) suggests that correlated log data, with technology to identify patterns, can be used to identify potential fraudulent activity at an early stage.

2.4.1 Unification of logs from different sources (correlation)

Typically in an investigation, logs from many sources will be brought together. Pasquinucci (2007) suggests that in order to correlate these logs automatically and efficiently the format of logs imported needs to be the same, with events classified in the same way and stored in a standard way (such as XML). Pasquinucci (2007) continues by suggesting that in reality this is difficult to achieve due to the vast array of log generating software already in use which uses propriety methods of logging data.

Schuster (2007) describes how the Unix Syslog operates in comparison to the Microsoft Windows NT and Microsoft Windows Vista formats. Syslog records both the constant and variable parts of an event record as a line of plain text in a log file. Windows NT stores the log information in a different way. Schuster (2007) explains that the constant information is held in a table, separately from the variable data. This has the advantage over Syslog as removing the constant data from the log file reduces the overall size of logs. Schuster (2007) suggests that the Microsoft Windows NT method required the entire log file to be loaded into memory and this caused a problem for servers with limited resources. Schuster (2007) then goes on to examine the new XML based format used by Microsoft Windows Vista, suggesting that the new method is more flexible and uses fewer resources than the techniques employed by Microsoft Windows NT. Schuster (2007) explains that the new format will pose difficulties for forensic investigators, stating that “without question the undocumented, proprietary binary XML format that Microsoft designed will be a major obstacle” (Schuster, 2007).

In order to overcome the problems caused by proprietary, often undocumented, log formats, the MITRE Corporation is coordinating work on Common Event Expression (CEE) which aims to become “the accepted way to describe and communicate events in log files” (The CEE Board, 2008).

The CEE Board proposes a new framework for event logging – producing standards for describing, storing and transporting log information. The CEE board argues that their framework will succeed because it takes in the wider issue of event logging rather than other frameworks which “only targeted a portion of the larger issue or were tied to individual vendors” (The CEE Board, 2008).

This standard for logging event data is still in development so other solutions are required meantime. Forte (2004) highlights the issue of log file correlation and suggests that unification is required:

> “Let us imagine, for example, an architecture in which we have to correlate events recorded by a website, by a network sniffer and by a proprietary

> application. The website will record the events in w3c format, the network sniffer in LibPcap format, while the proprietary application might record the events in a non-standard format. It is clear that unification is necessary here." (Forte, 2004).

In the context of log data from Intrusion Detection Systems (IDS), Livnat, Agutter, Moon, Erbacher and Foresti (2005) discuss how correlation by common attributes can overcome the problem of occasional false positives while highlighting real issues:

> "One approach to resolving these issues is to correlate various alerts by common attributes. This approach is based on the premise that while a false positive alert should not exhibit correlation to other alerts, a sustained attack will likely raise several alerts. Furthermore, real attack activities will most likely generate multiple alerts of different types." (Livnat et al., 2005).

In addition to the problem of logging event data in a consistent way there are issues surrounding the accuracy of timestamps, used to establish the order of events.

2.4.2 Time stamping

In order to correlate the log data for the production of timelines, the timestamp recorded by logs needs to be recorded in a constant and reliable manner. Gorge (2007) suggests that NTP (Network Time Protocol) may be used by system administrators. Forte (2004) has a different view about NTP suggesting that NTP is vulnerable and that in a distributed system a time stamping appliance can be used to handle the events. This appliance is synchronised with atomic clocks and provides a high degree of reliability.

2.5 Visualisation of log data

The issue of presenting log information is a serious one:

> "A great deal of time is wasted by analysts trying to interpret massive amounts of data that is not correlated or meaningful without high levels of patience and tolerance for error" (Teerlink & Erbacher, 2006).

Teerlink and Erbacher (2006) go on to suggest that tools such as EnCase and Helix do not currently provide advanced visualisation features in the process of data correlation and analysis.

Marty (2008, p.5) sums up the need for visualisation of log data, "a picture is worth a thousand log records". Marty (2008, p.80) considers that issues can occur during visualisation where by a trade off has to occur when deciding on how much data to show. In the case of a histogram, it is suggested that the time resolution is critical for being able to identify when events were logged, however too many bars become an issue if the resolution is too high.

Shahar, Goren-Bar, Boaz and Tahan (2005) developed KNAVE-II, a system which provides a graphical interface with time-lining capabilities for clinical data. Through their research it was proved that users were able to query the data more quickly and accurately than using conventional methods such as paper charts or Excel spreadsheets.

In addition to the issues of how visualisation can benefit a time-lining system over traditional text based systems, work has been done on how using colour in the design can improve visualisation by conveying extra information:

> "An effective design presents information in an organized manner, making it easy for the viewer to understand the roles and the relationships between the elements." (Stone, 2006, p.1)

Stone (2006) suggests that applying the correct principles, when using colour, is important in order to draw attention to the important elements and ensure that the message being presented to the end user is legible.

Shniederman (1996) has a mantra which describes the basic principles a design should follow for visualising data of "Overview first, zoom and filter, then details-on-demand". Shniederman (1996) suggests that the user of a system will first wish to see an overview of the data, then zoom in on an area of interest, filter out unwanted data and finally see the required details. He continues by suggesting that dynamic advanced filtering options allow the user to use OR and AND functions in order to filter the information as required.

2.6 Conclusions

From this literature review, the process by which digital forensic investigators find digital evidence, process it and present it in such a way as to establish timelines of the events surrounding an incident has become clear.

It is clear that automation significantly improves the ability of an investigator to gather data from a variety of log files and correlate log data from different systems which have different ways of storing their log data. There are efforts to standardise event logging, however current systems do not yet implement these standards, so log correlation is necessary to integrate logs from different systems, XML for instance can be used to store data in a common way. The issue of time stamping log data is a serious one and the current method used by many has involved a reliance on NTP. This has potential security vulnerabilities which can be overcome by having a secure, dedicated time stamping device. Once log data has been correlated, it is then possible to establish links between events and present this information in a timeline.

It has been suggested that visualisation, in particular time-lining, can vastly improve the efficiency of an investigator when looking through evidence, enabling them to spot potential issues and anomalies in log data. Users typically see an overview of the data before zooming, filtering and finally viewing specific details. The use of colour in visualisation is important as used correctly it can enhance the user experience and draw attention to important information. In another domain, it has been found that a bespoke visualisation tool out-performs paper-based data and Excel spreadsheets.

It is now possible to take forward the solutions found in the literature review with the goal of developing a prototype system which improves on existing event logging and time-lining systems. Specifically it will be possible to overcome log correlation difficulties by utilising a standard log format in XML, store log information centrally in a database and present time-lines of event patterns in a visual manner to the user of the prototype, such as a forensic investigator or system administrator. The issue of accurate time stamping, although undoubtedly important, will not be taken forward as it has been considered to be outwith the scope of this thesis which focuses mainly on the presentation of event information to the user.

3 Design and Methodology

3.1 Introduction

In order to produce a system for enhancing current methods of digital forensic investigation by standardising the way event information is recorded, by correlating log information and by improving presentation to the investigator, it is essential to have an informed and well thought out design. This chapter includes a high level design of the entire system and its individual components. The work carried out in Section 2 and the forthcoming People Activities Context Technologies (PACT) analysis will influence and guide the design decisions taken; these will be discussed during the design stage and later reflected upon during the evaluation.

By specifying tests for each component, and for the system as a whole, it will be possible to evaluate the efficiency and effectiveness of the design and of the implemented system which evolves from it.

3.2 PACT Analysis

In order to establish some principles around the design, it was decided that a PACT analysis would help with the decisions to be made around the user interface and monitoring components:

People - The system is likely to be of most use to forensic investigators, IT security teams and IT managers. It can safely be assumed that these people will have a good working knowledge of computing systems and the Windows operating system.

Activities - The activities undertaken will be monitoring certain user activities, collecting the logged data in an organised and quickly searchable way and displaying this log data in such a way that patterns and anomalies can be identified whilst still allowing the user to drill down into the details and establish the sequence of events.

Context - The system is to be used in the event of an investigation into activity carried out on machines. It can also be used as a monitoring device to highlight potential malicious activity.

Technologies - The system could be run on a variety of machines; laptops in the field, desktops in an office etc. Monitoring systems need to run on the machines under investigation. Log data will be stored centrally once imported into the system (database), this can be accessed from different systems running a user interface.

3.3 Choice of programming language

For the implementation of the system, a choice had to be made as to which programming language would be used. Given the timescale involved in order to complete the project it was clear that the options were Java, which had previously been studied, or Visual C# 2008 (C#) in conjunction with the .NET framework.

It has been decided that C# will best serve the project as the large quantity of class libraries available for interacting with the Windows operating system, Microsoft SQL Server platforms and support for the third party ZedGraph set of classes, for creating line and bar charts (ZedGraph, 2007), will be of great use for visualising log data. In theory the learning curve for switching to C# from Java (given previous experience with Java) will not be too steep as the two languages are similar in their syntax and operations. Additionally, the use of the Windows Presentation Foundation development platform means the latest methods can be used for developing the user interface and binding information to user interface objects.

3.4 Development method

It was decided that a rapid prototyping design and implementation method would be best suited to the project. According to UsabilityNet (2006), this enables a swift development phase and provides a demonstration system on which it is possible to perform both technical and user evaluation. Given the limited time available for implementation and the requirement of having a system on which to perform evaluation, rapid prototyping was chosen over paper based prototyping or a more involved implementation. In addition, such a rapidly developed prototype can be used as the basis for further development and the creation of a fully functional system.

3.5 Components

The nature of the system, determined from the PACT analysis, suggests that a modular approach to the design and implementation would be appropriate. A modular approach will allow the design to be implemented in stages with the aim of creating a more robust solution, each component being subject to testing before being included in the overall system. A modular approach will also allow for changes to be made to individual components (such as the database element), allowing for more detailed performance evaluation.

From the literature review, it is clear that in order to improve security and keep a centralised collection of data which can then be analysed, it is necessary to operate logging applications on client machines to gather the data which is then stored in a secure manner on a central server (the database). These applications should have a small memory footprint and will be run in the background so a console based interface is most appropriate. In order to ensure date and time consistency, all times will be converted to Coordinated Universal Time (UTC). This overcomes issues of daylight saving and multiple time zones.

The design, shown in Figure 1, covers each of the components which together make up the system. Each logging component will store its recorded events in a XML format in order to aid in the further processing and storage of the log data. Each XML log entry will incorporate a SHA-256 hash of the log data contained within the record. This will allow for checks to be made to ensure there is no data corruption and will highlight if any data has been modified in the log entry fields. SHA-256 was chosen over SHA-1 as the National Institute of Standards and Technology (2009) has indicated that SHA-1 is not sufficiently collision resistant and therefore new applications should use SHA-2 functions, such as SHA-256.

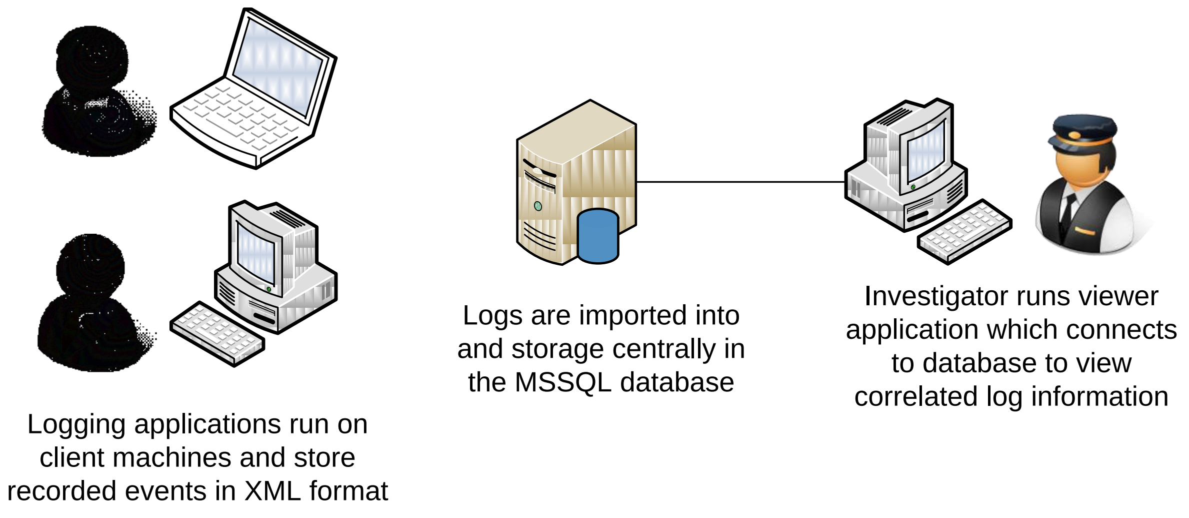

Logging applications run on client machines and store recorded events in XML format
Logs are imported into and storage centrally in the MSSQL database
Investigator runs viewer application which connects to database to view correlated log information

Figure 1: Overview of prototype system

3.5.1 The Database

The database component will provide a secure and efficient means of storing the entire log data collected from client machines. Given the scale of data and the security requirements, especially to prevent modification of data, it has been decided that the Microsoft solution of SQL Server 2008 will be used. This will also provide the means of linking in the data from clients and outputting data to a log viewer and visualisation component using the database classes and methods provided by the .NET framework.

In addition to the storage of the log data, the database component will also be responsible for the correlation and organisation of the data – such as using sort and searching capabilities, accessed through the Log and Timeline Viewer component.

An entity relationship diagram shown in Figure 2 highlights the relationships between the sets of data collected and how the data is to be organised within the database system.

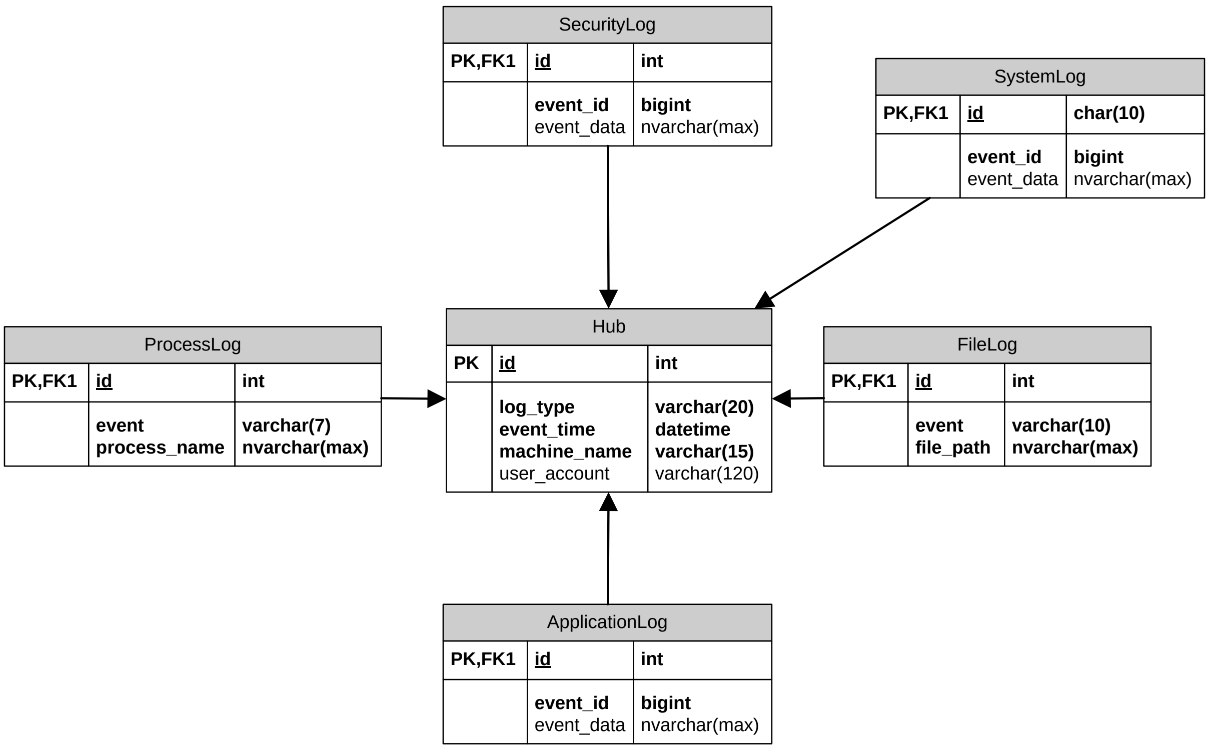


Figure 2: Database entity relationship diagram

Data types for each attribute have been based on the information to be stored. For example for storing the machine name a 15 character limit has been set. According to Microsoft (2009) 15 characters is the maximum length of a machine name. For the purposes of the prototype the machine name has been used to uniquely identify computers as in the controlled test environment no two machines will have the same name. In a larger scale system it could be possible that two or more machines would have the same name. Therefore a more complex solution would have to be devised such a hash from a series of variables possibly involving machine name, processor serial number and the MAC address of the network interface.

3.5.2 Windows Event Logs

Although the Windows Event Log service has been found to have security issues in previous studies, it can still provide useful information in the event of an investigation. A module will be dedicated for gathering the event log data and organising it for storage by the database.

The module will gather event data from the Application, System and Security logs from client machines of interest to the user of the system. Unlike the other modules concerned with collecting client machine data, the event log module will use the built in classes provided with the .NET framework to query the Windows Event Log service on client machines and gather in the data. As there is little control over the formatting of the event log data at the time of the client machine writing the logs this module will tidy the data and organise it, in a XML format, to be consistent with the other data to be held in the database.

3.5.3 File Watcher

The File Watcher module sits on the client system and logs various directory and file changes. These include file creation, deletion, renames and changes to file contents. By making use of the FileSystemWatcher class which is provided as part of the .NET framework, it will be possible to define specific folders and file types to monitor. A careful balance will have to be made to ensure malicious activity is logged without logging too many false positives which would increase the processing required to filter through log data and increase the storage requirements. For example the system could be configured to record the creation and deletion of .doc files within “My Documents” and the creation, deletion, renaming or changes to key system files (.dll, .cab, .exe) within the C:\Windows folder.

3.5.4 Process Logger

The Process Logger module records the starting and stopping of processes on the client machine. The process name and process ID are recorded in addition to the other information which is standard across the modules. The starting and stopping of the processes are time stamped so it will be possible, when viewing the correlated log data, to draw conclusions about the processes when in context with information gathered from the other logs. For example if the winword.exe (Microsoft Word) process started, then a .doc file was created and the winword.exe process stopped it would be possible to determine the program used and the username of the person that created the new .doc file which the File Watcher module logged.

3.5.5 Log and Timeline Viewer

In order to provide a clear and fast interface, the design of the Log and Timeline Viewer will be organised around three functions; importing and filtering data, log text viewer and visualisation. An initial sketch was made, shown Appendix 3.1, which will form the basis of the user interface for the Log and Timeline Viewer application.

The Log and Timeline Viewer component will aid in the correlation of log data from different sources, an issue discussed in the Section 2.4.1. While data to be logged has been designed with consistency across the client components, it will be necessary to do some processing on the data. The user will be able to import the log data into the database using the functions provided in the Log and Timeline Viewer. Clearly it will be essential that the original data is not modified when the logs are parsed and the data inserted into the database in order to ensure the system provides unaltered results to the investigator and, potentially, to a court of law.

Once the log data is held in the database, the Log and Timeline Viewer will also be responsible for allowing the digital investigator to browse through and sort the correlated log data in order to identify and investigate issues which have arisen. Visualisation will play a major role in this module as it allows the investigator to, at a glance, establish the level of log activity and identify anomalies. By using the ZedGraph classes the system will be able to display graph and pie charts of information quickly and support zoom and pan functions so the user can drill deeper into the results. This is of particular use when drawing histograms. As was discovered in the literature review, the time resolution is important for allowing the investigator to pinpoint when events were raised. ZedGraph's zoom functionality will allow the user to zoom into areas of interest and therefore instantly adjust the time resolution to be as accurate as possible without having to resort to reading the text log records. While most of the programming can make use of the latest technologies (C# Windows Presentation Foundation), it will be necessary to make some adjustment for the ZedGraph element as it has been designed for use with Windows Forms rather than being a native WPF library. At this stage in the development of the prototype there were no suitable charting libraries available which used WPF.

Although for the purposes of the prototype the Log Viewer will operate on the same physical machine as the database for practical and performance reasons, as the data is held separately, it would be possible for the system to be enhanced in future, for example by having a web version of the interface.

3.6 Testing and Experiment Design

This section discusses the design of the experiments, both technical and in relation to user feedback. The results from the experiments are documented in Section 5. To aid in the evaluation of the system, it is necessary to design testing methods which will allow conclusions to be made as to how the system functions in relation to the aim and objectives of the project and how the system responds to the issues found during the literature review.

3.6.1 Functionality Testing

In order to evaluate all aspects of the system to ensure they are operating as intended, the console applications will be used to record log data. It should then be possible to collate and store this data in the database component and view it in the Log and Timeline Viewer. The information displayed in the Viewer will be checked against the raw log data to ensure the original data has not been altered (an essential requirement set out in the design). If these tests are passed, it will show that the system operates with the functionality proposed at the design stage.

3.6.2 Technical Testing (impact on system, robustness and scalability)

A range of measurements can be taken in order to build a picture of system performance. The system will be run under a number of test conditions with varying quantities of data, placing different levels of demand on the system. System responsiveness will be recorded, by checking the user interface does not freeze whilst processing log data, as will factors such as memory usage and processing time. At the evaluation stage the results of this test can be discussed and it can be established whether or not the system can cope with the demands place upon it and remain efficient in its use of system (host computer) resources. If the prototype

application's resource usage is too high then it will interfere with the work of the user operating the computer being monitored.
In order to assess the impact the system has on the host system and how responsive the application remains when the scale of data processing increases, tests will be carried out. If the system is robust it will remain responsive and will not crash when the amount of data processing increases.

3.6.3 Experiment 1 – Time taken to load XML log data into the database

By measuring the time taken to load in log data and CPU utilisation, the impact on the system can be assessed. By initially testing without using threading and then repeating the tests using threading techniques it can be seen how the system's responsiveness is affected and if the use of threading improves system robustness. The test will first be run on the test machine (laptop with 4GB ram, Core 2 Duo 1.66GHz processor, Windows 7 Professional 64bit) and will allow the application to make full use of the processing power available. The test will then be repeated with processor maximum state set to 50% (set using the operating system's power management features). This will highlight the performance difference one could expect between running the system on a reasonably powerful PC and one that is perhaps older or using a less powerful mobile CPU.

3.6.4 Experiment 2 – Time taken to present chart and grid data

Investigating the time taken for the system to present the visual information, and assessing the appearance of this information, will help assess how scalable the system is. If the time taken to draw charts becomes too long then the system's scalability will become an issue. Additionally, if charting information becomes unreadable as the number of points increases, then the system's scalability will come into question. Marty (2008, p.110) suggests that for bar charts and line graphs a maximum of 50 points can be reasonably displayed. This test will see if this theory applies to the graphing solutions implemented within the viewer application.

3.6.5 Experiment 3 – Memory usage

By testing the memory usage of the system, it will be possible to assess whether or not the system's memory usage remains stable and within a level which would indicate that the system can run on a number of systems which may not have large amounts of memory available (laptops in the field, tablet PCs etc). It is expected that memory usage will increase when the system is performing more intensive tasks, however it should not get so great that the system runs out of RAM memory and begins paging to disk.

3.6.6 Experiment 4 – System robustness with incorrect user actions

In order to specifically test the system's robustness to incorrect user actions a series of situations will be played out and the results recorded. If the system is able to recover, with a useful error message and with data integrity remaining untouched in addition to the user interface not crashing, then this would indicate that the system remains robust.

3.6.7 User feedback and evaluation

As discovered during the literature review, it was found by Shahar et al. (2005) that their system performed better than traditional approaches of looking at patient data. Following the same theme, each user will be asked to complete a questionnaire asking them about their preferences between systems for displaying log and timeline information.

It is expected that users will prefer the prototype implementation as it provides visualisation features not found in text files or the Microsoft Windows Event Viewer.
In addition to questions specific to visualisation, users will be asked to rate how useful or otherwise they find specific features of the prototype in order to determine if the design meets the objective of improving on existing systems. One question in particular will focus on how long the user would be prepared to wait for log data to be imported into the database. It is expected that users would be prepared to wait in excess of 10 minutes, particularly users who have experience of other analysis systems.

3.7 Conclusions

The prototype solution has been designed to incorporate the features required which have been identified though the literature review and a PACT analysis. The solution will be split into a main Log and Timeline Viewer application (importing, filtering and displaying data) and three smaller console based applications which gather in Windows Event Logs, process and file information. The console applications record event information in XML files which are then processed and the information is imported into a database which centralises all log information.

The following section details how the implementation followed the design and contains code snippets to further explain how features were developed.

4 Prototype Implementation

4.1 Introduction

The purpose of this section is to give an overview of how the features which were designed in the previous section were implemented in the prototype, explanation of features which were changed and added since implementation was commenced and issues which arose or new ideas arrived at during the development of the prototype.

The goal of the implementation is to provide a framework on which to carry the technical and user evaluation. By implementing the features which have been designed, based on the conclusions of the research, it will be possible, through experimentation and evaluation, to determine whether the prototype system achieves the goal of this thesis which is to produce a system which enhances current technologies.

4.2 Event Logging Applications

4.2.1 Common functions

Each of the logging applications were implemented as console applications. By running as a console application the logger consumes fewer resources than a graphical application, which could be important on a client system being monitored. Each of the logging applications stores its log information as an XML file which shares a common set of XML elements to describe a record, event time, machine name, username event type and checksum.

The XML element names were chosen based on the "Best Naming Practices" by W3Schools (2009). The checksum element comprises of a SHA-256 hash of the data held within the other elements.

4.2.2 Window Event Log Reader

As stated in Section 3.5.2, the event reader application runs on the system under investigation, gathers in the Windows event logs and converts them into an XML format reader to be loaded in the database component. The code in Figure 3 demonstrates how a loop was used to read in each event record, accessed by means of the EventLogEntry class which is provided with the .NET Framework.

```
foreach (EventLogEntry entry in aLog.Entries)
                    {
                        event_data = entry.Message;
...
}
```

Figure 3: Reading event log records

Whilst implementing the system, it was found that writing to the XML file was slow. As originally implemented, the XML file was opened, written to and saved every time a new event was loaded into the system from the event logs held by the operating system. In order to improve performance a small piece of logic was added, see Figure 4, which meant the system only had to create, open and save the XML file.

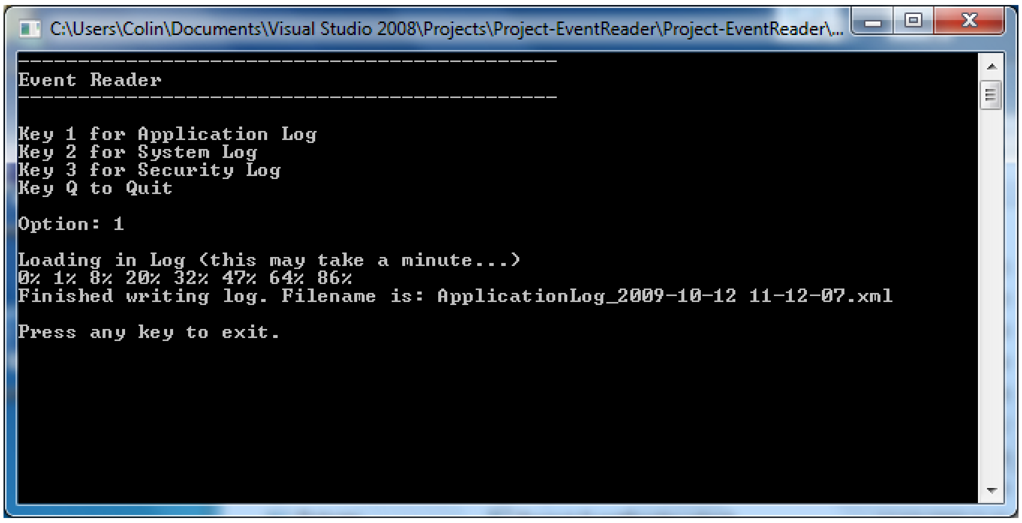
Additionally, as the system takes time to load in the records and write out to the XML file, a method of informing the user of the system as to progress was created. The code for this can be seen in Figure 5. This code provides the user with an update once a second as to the percentage of load records which have been processed so far. It was found during implementation that updating the progress percentage more than once a second caused the console window to flicker, whilst updating less often meant gave the impression to the user that the system was slower than was in fact the case. The screenshot, shown in Figure 6, depicts a typical user action where the user has chosen option 1 to read in and convert the machines Application event data to the XML format used by the prototype. As can been seen in Figure 6, the user has been given feedback as to the progress being made by means of percentage completed. Once the process has been completed, the user has been informed of the file name of the XML file.

```
if (count == total)
      {
      last = true;
      }

//write record code

if (last == true)
      {
      xmlDoc.Save(filename);
      }
```

Figure 4: Saving XML code

```
if (was.AddSeconds(1.00) < DateTime.Now || count == 1) //update on first
loop then update once a second, stops output flickering
      {
            percentage = (count *100) / total;
            Console.Write(percentage.ToString() + "% ");
            was = DateTime.Now;
      }
```

Figure 5: Loading progress output code


C:\Users\Colin\Documents\Visual Studio 2008\Projects\Project-EventReader\Project-EventReader\...

```
---------------------------------------------
Event Reader
---------------------------------------------

Key 1 for Application Log
Key 2 for System Log
Key 3 for Security Log
Key Q to Quit

Option: 1

Loading in Log (this may take a minute...)
0% 1% 8% 20% 32% 47% 64% 86%
Finished writing log. Filename is: ApplicationLog_2009-10-12 11-12-07.xml

Press any key to exit.
```

Figure 6: Event Reader Application screenshot

The XML file created by the event log reader places each record with a record element (recorded_event) which contains elements for each piece of data held within the records (fields in the database). Figure 7 is an example of a XML record as recorded in the XML file by the system.

```
<recorded_event>
      <event_time>2009-8-29 20:36:36</event_time>
      <machine_name>Dell-7</machine_name>
      <user_account>Not Recorded</user_account>
      <event_id>1003</event_id>
      <event_data>The Windows Search Service started.</event_data>
      <checksum>0D-3D-0D-F1-53-47-27-BF-A2-F3-A8-DA-35-B4-D0-CB-F0-50-A6-
      B4</checksum>
</recorded_event>
```

Figure 7: Example system log event record (XML)

File names written as the format LogType_year-month-day hh-mm-ss.xml. This format ensures that each file name is unique and can easily be understood by the user when they need to find it again for loading into the database component.

It was discovered during the implementation, that the application would crash when trying to read in the host machine’s security event log. This was due to the security restrictions imposed by the Microsoft Windows Vista and Windows 7 operating systems. By running the application with administrator privileges, this problem was overcome as the application then had sufficient rights to access the security log information. Whilst it is not ideal that an application has to run with elevated privileges, as this may introduce other security problems, this issue demonstrates that the latest Microsoft Windows operating systems are designed to protect log data from unauthorised access.

4.2.3 File Watcher

As described in the design overview, the file watcher application makes use of the File System Watcher class which comes with the Microsoft .NET framework. The application was implemented using two classes. The first class describes the file path to be watched (in this case temporary internet files), creates a new FileWatcher object (second class) and handles messages to be displayed in the console window. This initial setting up of the FileWatcher is shown in Figure 8.

```
string username = System.Environment.GetEnvironmentVariable("USERNAME");
//ensures file watcher will watch current users files rather than hard
coded
string path =
"C:\\Users\\"+username+"\\AppData\\Local\\Microsoft\\Windows\\Temporary
Internet Files";
FileWatcher doIt = new FileWatcher(path);
Console.WriteLine("File Watcher is Running");
Console.WriteLine("Path being watched: " + path);
Console.ReadLine();
```

Figure 8: Setting up the FileWatcher code

The second class makes use of the FileSystemWatcher to monitor the designated file path for changes as shown in Figure 9. When a change is detected, the method which corresponds to the type of change (creation, rename, deletion or change) is run which in turn runs the WriteXML method which logs the event information to the XML file which is outputted by the application.

```
public FileWatcher(string path)
{
      string directory = path;
      FileSystemWatcher WatchFile = new FileSystemWatcher(directory);
      WatchFile.Created += new FileSystemEventHandler(FileCreated);
      WatchFile.Renamed += new RenamedEventHandler(FileReNamed);
      WatchFile.Deleted += new FileSystemEventHandler(FileDeleted);
      WatchFile.Changed += new FileSystemEventHandler(FileChanged);
      WatchFile.EnableRaisingEvents = true;
      WatchFile.IncludeSubdirectories = true;
}
public void FileCreated(object sender, FileSystemEventArgs e)
{
      WriteXML(e.ChangeType.ToString(), e.FullPath);
}

...
```

Figure 9: Capturing events code

As with the event reader application, the file watcher application writes out the log information to a XML file. In this case the XML file is created when the first event is to be logged. The event is written to the file, saved and closed. When another event is to be logged, the file is reopened and the new event is appended. This ensures all the events are logged and saved should the file watcher application quit unexpectedly (such as if the machine is powered off). The XML schema remains similar in that each event's details are recorded within a recorded_event element, an example of which is recorded in Figure 10.

```
<recorded_event>
      <event_time>2009-11-20 17:37:32</event_time>
      <machine_name>DELL-7</machine_name>
      <user_account>DELL-7\Colin</user_account>
      <event>Created</event>
      <file_path>C:\testfiles\New Text Document.txt</file_path>
      <checksum>9C-5F-D9-59-94-5D-E6-CE-16-F1-BB-B9-8E-A5-73-2A-B9-83-D3-
      F9-44-02-8B-36-70-FB-06-F0-7D-89-5F-70</checksum>
</recorded_event>
```

Figure 10: Example file watcher event record (XML)

4.2.4 Process Logger

The process logger works in a similar way to the file watcher application in that it is a console application which monitors activities on the system and logs any events to an XML log file. In this case, the events to be logged are the starting and stopping of processes on the system being monitored. The code for monitoring the processes loops once a second in order to identify any processes which have started or stopped since the last loop was run through. As with file watcher application, a separate WriteXML method is used to write each event out to an XML file which is named with the date and time, in UTC format.

Originally, the intention had been to identify and record the account name which started or stopped a process. During implementation it was found that, although technically achievable using Windows Management Instrumentation (WMI), this method was too slow which could cause the loop to take more than a second to finish and thus potentially result in new process events not being logged correctly. A compromise was found by logging the account name of the user logged into the system at the time of the processes events being logged.

An example event record for showing logged process information is shown in Figure 11.

```
<recorded_event>
      <event_time>2009-11-20 17:30:03</event_time>
      <machine_name>DELL-7</machine_name>
      <user_account>DELL-7\Colin</user_account>
      <event>started</event>
      <process_name>firefox</process_name>
      <checksum>96-2F-6B-F8-59-16-B5-E0-25-66-FA-5A-28-CF-04-AD-19-21-EC-
      8D-28-C9-11-32-F6-A9-7B-E5-94-7E-4A-EA</checksum>
</recorded_event>
```

Figure 11: Example process event record (XML)

4.2.5 Log and Timeline Viewer

The Log and Timeline Viewer is the main interface of the system allowing the investigator to import records, filter and display event data in order to understand the sequencing of events. The user interface was designed around three tabs; dashboard, load logs and explore data. As this component of the prototype was developed using the latest programming methods, the user interface design and the code behind are kept separate. The user interface is implemented in Extensible Application Markup Language (XAML) which, according to Microsoft (2008), simplifies the creation of UI elements and separates the UI from the run time code. The more traditional C# code behind handles all the user interaction events, connections to the database and so on. This solution has similarities with webpage design where by the styling is defined in a Cascading Style Sheet and the content is stored in HTML. The dashboard tab provides the user with an overall view of the system's contents, with a pie chart used to represent the proportion of log records held within the database. Figure 12 depicts the four main classes and the methods contained within them. Detailed code listings for these classes are presented in Appendix 3.2.4.

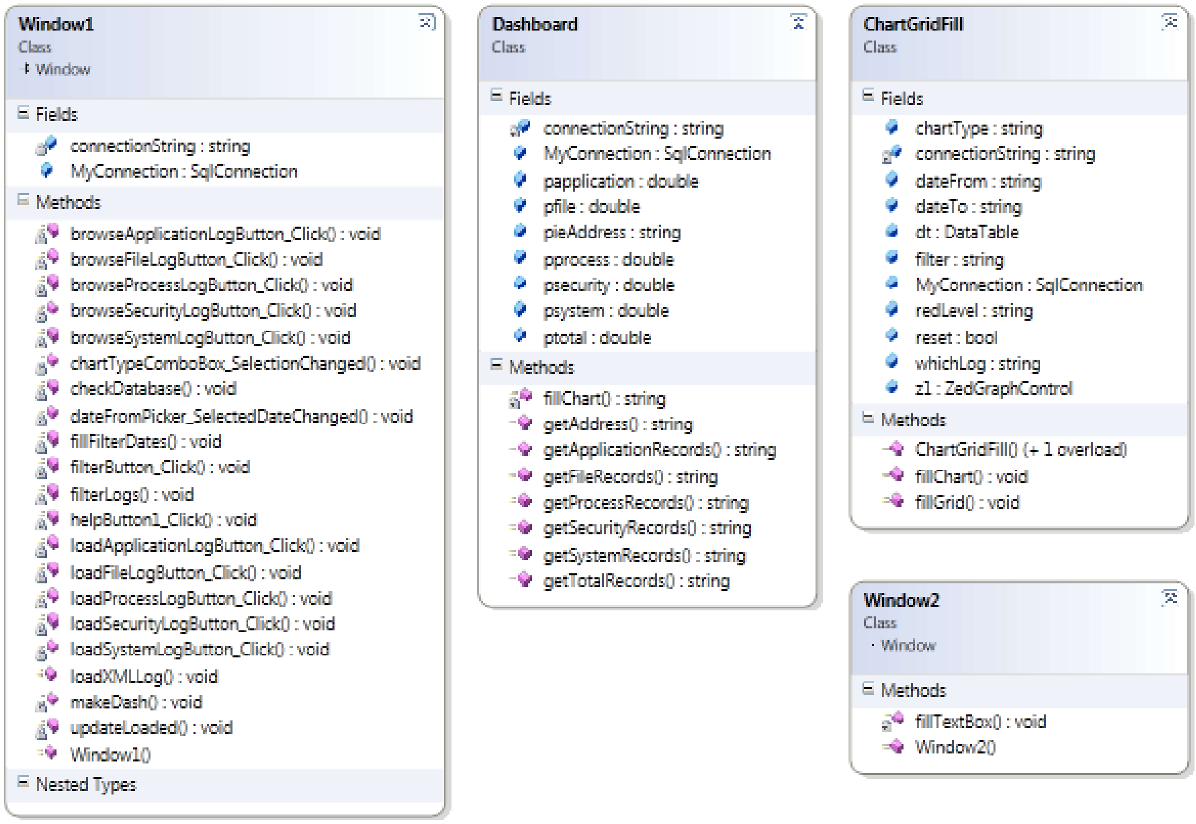

Window1
Class
Window
Fields
connectionString : string
MyConnection : SqlConnection
Methods
browseApplicationLogButton_Click() : void
browseFileLogButton_Click() : void
browseProcessLogButton_Click() : void
browseSecurityLogButton_Click() : void
browseSystemLogButton_Click() : void
chartTypeComboBox_SelectionChanged() : void
checkDatabase() : void
dateFromPicker_SelectedDateChanged() : void
fillFilterDates() : void
filterButton_Click() : void
filterLogs() : void
helpButton1_Click() : void
loadApplicationLogButton_Click() : void
loadFileLogButton_Click() : void
loadProcessLogButton_Click() : void
loadSecurityLogButton_Click() : void
loadSystemLogButton_Click() : void
loadXMLLog() : void
makeDash() : void
updateLoaded() : void
Window1()
Nested Types
Dashboard
Class
Fields
connectionString : string
MyConnection : SqlConnection
papplication : double
pfile : double
pieAddress : string
pprocess : double
psecurity : double
psystem : double
ptotal : double
Methods
fillChart() : string
getAddress() : string
getApplicationRecords() : string
getFileRecords() : string
getProcessRecords() : string
getSecurityRecords() : string
getSystemRecords() : string
getTotalRecords() : string
ChartGridFill
Class
Fields
chartType : string
connectionString : string
dateFrom : string
dateTo : string
dt : DataTable
filter : string
MyConnection : SqlConnection
redLevel : string
reset : bool
whichLog : string
z1 : ZedGraphControl
Methods
ChartGridFill() (+ 1 overload)
fillChart() : void
fillGrid() : void
Window2
Class
Window
Methods
fillTextBox() : void
Window2()

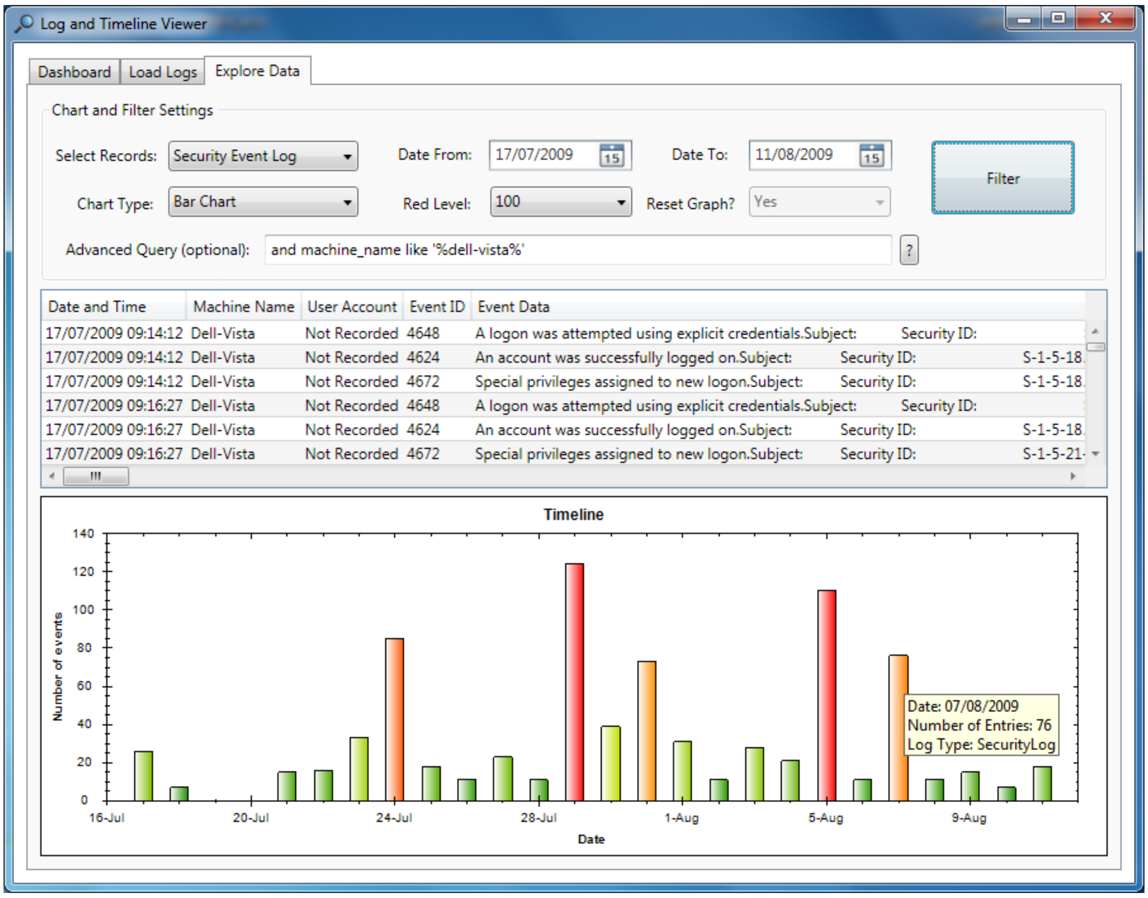
Figure 12: Class Diagram of viewer application

The load logs tab provides a means for the user to load the XML files created by the console applications on machines under investigation into the database. Given the time taken to load in log files, threading was used to ensure the user interface thread did not become unresponsive whilst loading in log data. The code in Figure 13 demonstrates how this was achieved.

```
private void loadApplicationLogButton_Click(object sender, RoutedEventArgs
e)
        {
            loadApplicationLogButton.Visibility = Visibility.Hidden;
            string latest = loadApplicationLogTextBox.Text;

            Thread thread_run = new Thread(delegate()
            {
                loadXMLLog(latest, "applicationlog");
            });
            thread_run.IsBackground = true;
            thread_run.Start();
        }
```

Figure 13: Use of threading code

The explore data tab includes the filtering, log text viewer and visualisation discussed in the design. Figure 14 is a screenshot of the log and timeline viewer with the explore data tab open.



Select Records: Security Event Log
Date From: 17/07/2009
Date To: 11/08/2009
Chart Type: Bar Chart
Red Level: 100
Reset Graph? Yes
Advanced Query (optional): and machine_name like '%dell-vista%'

Date and Time	Machine Name	User Account	Event ID	Event Data
17/07/2009 09:14:12	Dell-Vista	Not Recorded	4648	A logon was attempted using explicit credentials.Subject: Security ID:
17/07/2009 09:14:12	Dell-Vista	Not Recorded	4624	An account was successfully logged on.Subject: Security ID: S-1-5-18
17/07/2009 09:14:12	Dell-Vista	Not Recorded	4672	Special privileges assigned to new logon.Subject: Security ID: S-1-5-18
17/07/2009 09:16:27	Dell-Vista	Not Recorded	4648	A logon was attempted using explicit credentials.Subject: Security ID:
17/07/2009 09:16:27	Dell-Vista	Not Recorded	4624	An account was successfully logged on.Subject: Security ID: S-1-5-18
17/07/2009 09:16:27	Dell-Vista	Not Recorded	4672	Special privileges assigned to new logon.Subject: Security ID: S-1-5-21-

Date: 07/08/2009
Number of Entries: 76
Log Type: SecurityLog

Figure 14: Log and Timeline Viewer screenshot

The system takes in the filtering options set by the user then runs two SQL queries on the MSSQL database (one for the data grid and another for the visualisation) and subsequently produces the results below the filtering settings.

The data grid element makes use of WPFs data binding support. This means that as the user runs queries on the database (through the filtering options) the results are stored in a data table which is bound to the data grid user interface element. This is an efficient way of programming as WPF handles adding each row of data into the grid and is very quick to update when a new query is run. The XAML code for defining the data grid and its attributes is shown in Figure 15.

```
<dg:DataGrid Margin="6,157,6,296" Name="logDataGrid"
ItemsSource="{Binding}" AutoGenerateColumns="True" IsReadOnly="True"
VerticalGridLinesBrush="LightGray" HorizontalGridLinesBrush="LightGray"
IsEnabled="True" IsHitTestVisible="True" HeadersVisibility="Column"
GridLinesVisibility="Horizontal" AlternationCount="0"
Background="#FFF0F0F0" RowBackground="White"
AlternatingRowBackground="WhiteSmoke" IsTextSearchEnabled="True" />
```

Figure 15: XAML defining the data grid

As described in Section 3.5.5, the visualisation is provided by the ZedGraph library. Bar, line and scatter charts are supported by ZedGraph and have been implemented in the prototype. This support for multiple chart types makes it possible to allow the user of the system to select their preferred chart type and when using the line chart option, to compare multiple query results on one chart. During the implementation it was found that ZedGraph could support colouring points/bars on a chart based on their value. By adding in a “red level” option to the filtering settings, it was possible to add a feature which colours charts red where points are above the set limit selected by the user. Adding this feature brings the prototype into line with the principles of using colour in visualisations described by Stone (2006). Figure 16 shows the C# code used to create a line chart with traffic light colours for points. During the implementation stage, Microsoft released an updated version on the WPF Toolkit which provided additional libraries for use with WPF which included chart controls. It was considered that switching from ZedGraph to the WPF Toolkit for the visualisation was unfeasible given the research which would be required into the operation of the WPF chart controls and the limited time available for the project.

```
LineItem myItem;
            if (reset == false)
            {
                myItem = myPane.AddCurve(whichLog + " " + filter, list,
System.Drawing.Color.Orange);
            }
            else
            {
                myItem = myPane.AddCurve(whichLog + " " + filter, list,
System.Drawing.Color.Blue);
            }
            //colour items based on value
            System.Drawing.Color[] colors = { System.Drawing.Color.Green,
System.Drawing.Color.Yellow, System.Drawing.Color.Red };
            myItem.Symbol.Fill = new Fill(colors);
            myItem.Symbol.Fill.Type = FillType.GradientByY;
            myItem.Symbol.Fill.RangeMin = 0;
            myItem.Symbol.Fill.RangeMax = Int32.Parse(redLevel);
```

Figure 16: Code defining a ZedGraph Line Chart

4.3 Conclusions

The system has been implemented as per the design. This has involved creating the three console applications which collect Windows Event Log, file and process event data. In addition the implementation has involved the setting up of a MSSQL database to bring together event records from different sources and constructing the main viewer application which enables the end user of the system to interact with the stored information and view timelines of events in a graphical manner. During implementation an additional feature, suggested by the literature, where colour is used to highlight high levels in the charts, has been added. The prototype, produced by following a rapid prototyping design and implementation technique, was now capable of being used as a framework on which to carry out technical testing and user feedback evaluation.

5 Testing Execution and Results

5.1 Introduction

Both technical and user testing was carried out on the prototype system as per the design in Section 3.6. The first part of this section, details the technical testing which was carried out in order to assess the performance and robustness of the system. The following part, 5.2, discusses how the user testing was implemented. By performing testing and experimentation on the system, it was then possible to evaluate the system as discussed in Section 6, from both technical and user perspectives as per the goals of the thesis which include evaluating the prototype in order to establish the extent to which the features implemented in the prototype improve upon current methods.

A set of test data was created and, to minimise the effect of outside factors, all unrelated applications and services were switched off on the test machine (where possible). Each experiment was carried out three times and the average (mean) result recorded. When carrying out the experiments it was found that the results from each test run were very similar, therefore it was decided that three runs were sufficient. The Performance Monitor management tool built into Windows 7 was used to measure memory usage and CPU utilisation for the processes being tested.

5.1.1 Initial testing - functionality

The initial testing was based around answering 3 questions:

- Can the console application read events logs, capture file and process activity and store this information in XML format?
- Can this data be imported into the database and viewed within the viewer program both as log data within a data grid and visualised using the ZedGraph system?
- Does the system alter the log data in any way (except changing formatting)?

Having tested each of the console applications, it was found that there were some issues with the processing logging application being unable to monitor process activity and write captured events to the XML log file. After some further investigation of the exceptions being thrown by the application, it was discovered that the Windows Account Control features of the host operating system (Microsoft Windows 7 Professional Edition) were preventing the processing logging application from being able to monitor processes. This was resolved by setting the console application to run with administrator privileges.

After the initial problems were overcome, it was found that the console applications read the event logs and captured the file and process activity as required.

Once it was confirmed that the console applications were able to function as required, the main application (Log and Timeline Viewer) was opened. It was found that it was possible to load the XML log data correctly into the system and that the log information was viewable in the data grid and as charts as per the design.
Having loaded XML log data into the database, SQL queries were run on the database to pull out all the records. The data in these records was compared to the data held within the XML logs. It was found that the data was not modified (apart from formatting). It was important that this was the case as it is fundamental to all event logging and digital forensic tools.

5.1.2 Experiment 1 – Time taken to load XML log data into the database

This experiment was repeated three times with the CPU state set to 100%. The figures produced were then averaged (mean). The experiment was then repeated with threading switch off in order to observe the program's behaviour when the loading in of the log ran in the same thread as the user interface. The results are shown in Table 1 and Table 2. The experiment was run 3 times again with the maximum CPU utilisation set to 50%. The results with the CPU set at 50%, both with and without threading, are shown in Table 3 and Table 4.

From the results, it is clear that size of the log file has a considerable impact on system performance. As the file size increases above 2MB the system takes a considerable time to load in the data. Additionally, if threading is not used then the user interface becomes unresponsive with load files over 1MB in size. It would appear to a user that the application had crashed.

Size of XML file	**Time to load into database (seconds)**	**Average CPU Utilisation (% of available)**	**Peak CPU Utilisation (% of available)**
10KB	0.1	20.2	32.7
100KB	0.3	25.9	31.6
1MB	3.8	32.1	46.2
2MB	11.7	40.4	73.2
10MB	235.3	87.2	99.5
20MB	924.7	95.1	100.0

Table 1: Time to load into database results. Full CPU utilisation.

Size of XML file	Crashed without threading?	Crashed with threading?
10KB	No	No
100KB	No	No
1MB	No	No
2MB	Yes	No
10MB	Yes	No
20MB	Yes	No

Table 2: Impact of threading results. Full CPU utilisation.

Size of XML file	Time to load into database (seconds)	Average CPU Utilisation (% of available)	Peak CPU Utilisation (% of available)
10KB	0.2	29.2	31.1
100KB	0.5	32.5	38.4
1MB	4.5	60.4	72.8
2MB	15.5	81.1	87.5
10MB	336.5	91.1	99.7
20MB	1355.2	96.2	100.0

Table 3: Time to load into database results. 50% CPU utilisation.

Size of XML file	Crashed without threading?	Crashed with threading?
10KB	No	No
100KB	No	No
1MB	Yes	No
2MB	Yes	No
10MB	Yes	No
20MB	Yes	No

Table 4: Impact of threading results. 50% CPU utilisation.

5.1.3 Experiment 2 – Time taken to present chart and grid data

As previously discussed, by measuring the time taken for the viewer application to display results to the user, it can be established whether or not the system performs quickly enough when the scale of log information increases. In addition, it can be seen whether or not the appearance of the visualisation feature remains acceptable as the period of log information increases.

The test data for this experiment consisted of log data gathered from three typical machines over a three month period. The data was duplicated, and timestamps adjusted, to represent data over a longer period. The log data representing two years was held in the database for all the parts of the experiment and the filtering options used to select the date period required.

As with the previous experiment, the average time taken was recorded and can be seen in Table 5. Figure 17, Figure 18 and Figure 19 show how the appearance of a timeline displayed as a bar chart changes as the time frame increases. As can be seen from the results, the visualisation element becomes more difficult to read when six months of data is presented. When a year's worth of data is presented in a chart it becomes too compacted and it is not possible to read individual points without making use of the zoom facility.

Whilst the charting element shows an increase in time to display results, it is the data grid which shows the most significant increase as the amount of log data increases and could indicate that the system would not be scalable in this respect.

Date Period	**Chart Type**	**Appearance (clear/passable/poor)**	**Time taken to present chart (ms)**	**Time taken to present grid (ms)**
1 Week	Bar	Clear	29	14
	Line	Clear	29	9
	Scatter	Clear	29	12
2 Weeks	Bar	Clear	30	16
	Line	Clear	31	15
	Scatter	Clear	32	17
1 Month	Bar	Clear	50	27
	Line	Clear	41	29
	Scatter	Clear	40	31
3 Months	Bar	Clear	64	67
	Line	Clear	63	70
	Scatter	Clear	61	69
6 Months	Bar	Passable	91	160
	Line	Passable	88	165
	Scatter	Clear	76	164
1 Year	Bar	Poor	135	621
	Line	Poor	130	619
	Scatter	Poor	119	631
2 Years	Bar	Poor	214	1121
	Line	Poor	212	1122
	Scatter	Poor	201	1125

Table 5: Appearance of chart and time to present data

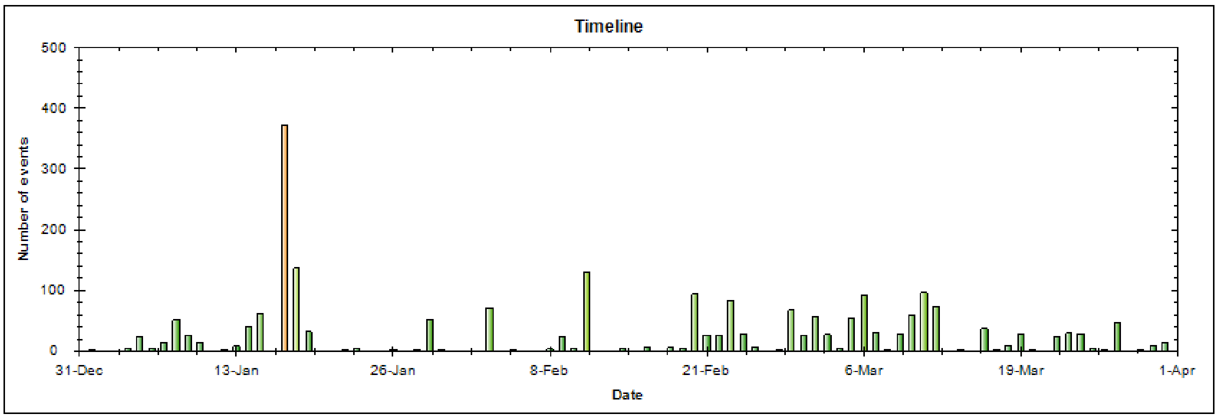


Figure 17: Bar chart - 3 months

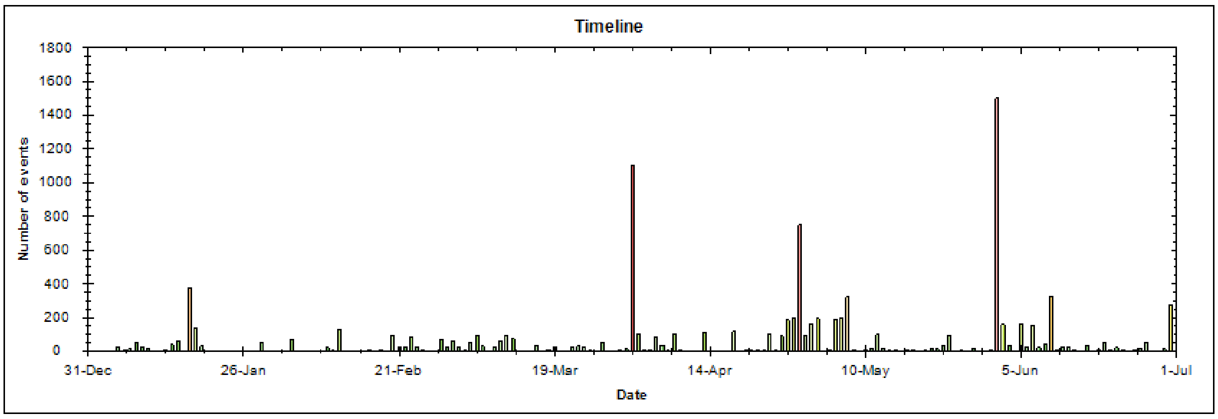


Figure 18: Bar chart - 6 months

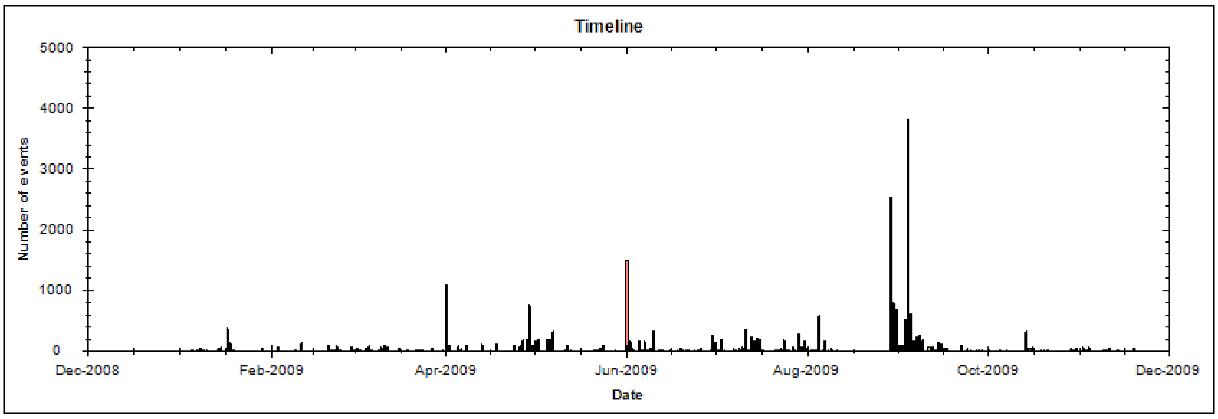


Figure 19: Bar chart - 12 months

5.1.4 Experiment 3 – Memory usage

By testing the memory usage of the system, it was possible to determine the likely impact the components would have on their host system. The testing of the viewer application involved monitoring the peak memory usage of the process while a number of typical operations were carried out. The results are displayed in Table 6. In order to test the memory usage of the console applications, it was necessary to run the application in the kind of environment in which they are expected to run. The results of the experiments into the memory usage of the console applications are displayed in Table 7. The Event Log Reader was tested whilst it was loading in application log data and writing out the log data to an XML file. In order to test the memory usage of the File Watcher application, an additional test application was created. The test application created a new file once a second for a total of a minute into a directory. The File Watcher was set to monitor this directory and log all the events. The Process Logger application was tested over a 10 minute period whereby a selection of processes were started and stopped.

Condition	**Peak Memory Usage (working set, MB)**
Application loaded with 20000 records	104
Filtered logs with 6 months data graphed	132
Filtered logs with 12 months data graphed	143
Loading in 1MB log	166

Table 6: Memory usage of viewer application

Application	**Peak Memory Usage (working set, MB)**
Event Log Reader	30
File Watcher	16
Process Logger	19

Table 7: Memory usage of log gathering applications

From the results, it can be determined that the viewer application requires significantly more memory than the logging applications. This is to be expected as the viewer had a graphical user interface and carries out more intensive processing than the console applications which are restricted to a pre-defined function. It can be seen that the console applications do not require a large amount of memory to operate which would suggest that, if run in the background, the user of a system being monitored would not be aware of the logging taking place. Whilst a memory usage of 19-20 MB would not be acceptable for a very lightweight system (PDA or mobile phone) the application could run on low powered portable machines such as tablet PCs and net books.

5.1.5 Experiment 4 – System robustness with incorrect user actions

The purpose of this experiment was to determine if the viewer application could cope with incorrect user actions. The system was subjected to a number of incorrect actions which are detailed with the results in Table 8. From the results it can be seen that the viewer application can cope with incorrect user actions relating to the filtering and display of the log data held within the database. It can also be concluded that there are issues surrounding the way in which the user loads log data into the system

– with the potential for duplicate or incomplete log data to be entered into the database.

Scenario	**Result**
User tries to select date range out with the range of data held by the system	If using date selector user cannot select wrong date. If manually entered system remains stable. **PASS**.
User enters advanced query with incorrect syntax	User is presented with a warning message. Data stored remains unaffected. **PASS**.
User tries to load a process log file using the application log load function	User is presented with an error message. System remains stable. **PASS**.
User loads in the same log file twice	Data is stored twice in the database. No error message is displayed. **FAIL**.
User closes application while loading in log data from XML file	System closes with no warning message. Only part of the log file data is stored in the database. **FAIL**.

Table 8: Robustness test results

5.2 Implementation and findings of user feedback

A questionnaire was devised to record user opinion on the design and functionality of the prototype system as a whole and establish the users' views on how the prototype Log and Timeline Viewer compares to other methods of storing and displaying log information.

To ensure consistency between feedback sessions, each user was given a brief demonstration of how each aspect of the prototype system could be operated before being given the opportunity to use the system themselves, whilst answering the questions set in the questionnaire. A total of seven users took part in the user testing, each with a background in IT, but with different levels of experience in the use of log management and digital forensic systems.

The questionnaire was split into two sections. The first was concerned solely with the operation of the prototype system. The second section, with the aid of a support sheet, was aimed at providing a means of comparing the prototype system with text files and the Microsoft Windows Event Viewer. In addition, the second section asked more general questions as to whether or not colour improved a user's ability to spot anomalies in bar chart data and which type of chart best displayed data over a 24 hour period.

The main results from the user feedback responses are documented below. For a complete set of responses and a copy of the support sheet please see Appendices 2.1 and 2.2 respectively.

The first two questions asked users to indicate how easy they found it to familiarise themselves with the prototype system and how easy they found it to load XML data into the database respectively. Six out of seven users indicated that they found it easy to familiarise themselves with the system, the seventh user described it as neither easy nor hard. All those taking part in the feedback sessions indicated that they found loading in the XML data to be easy.

There was a more mixed response in relation to the question as to how useful users found the dashboard; however no users chose to describe either feature as useless. Six out of seven users described the advanced query option as being useful. The seventh user chose to describe the feature as neither useful nor useless.

The final question in the first section asked users to select how long they would be prepared to wait for a large amount of log data to be loaded into the database. The answers given to this question were wide ranging. Two users said they would wait less than five minutes, two said they would wait five to ten minutes and the remaining users said they would wait more than ten minutes. In addition to answering the question specifically two users commented that providing the user with more feedback on the process, such as via a progress bar, would be beneficial.

The first set of questions in the second section asked the user to rate from a scale of one to three, different systems, in order of preference with three being best. The questions, and ratings given by users and added together, are displayed in Table 9. From the results it can clearly be seen that the prototype system was considered to be better that text files or the Microsoft Windows Event Viewer for getting an overview of log data and how filtering the data. The Microsoft Windows Event Viewer fared better with regard to viewing detailed log information. Possible reasons for this are discussed in the forthcoming Evaluation Section, 6.2.1.

Question	**Text File**	**Event Viewer**	**Prototype**
Indicate your preference regarding getting an overview of event data.	7	14	21
Indicate your preference regarding filtering log data.	7	16	19
Indicate your preference for viewing detailed log data.	10	19	13

Table 9: User feedback ratings. Comparing systems.

When asked to decide on whether the use of a traffic light colour scheme on a bar chart improved their ability to spot anomalies there was a clear trend in responses. Six out of seven uses preferred the graph with colour. The remaining user found it difficult to distinguish between the colours used on chart and so chose not to answer the question. The difficulties experienced by this user are further explored in the Evaluation Section, 6.2.1.

The last question asked of users, before being given an opportunity to give general comments, asked them to indicate their preference of graph type based on a one to three rating with three being the best. The data shown in the charts was of user login data over a 24 hour period, similar in nature to the log data displayed by the prototype system. The results were very clear, in that the responses were identical. The bar chart was the most preferred option, followed by a line chart, with the pie chart being the least preferred option.

5.3 Conclusions

Testing has been carried out from both technical and user perspectives. It has been found that the prototype supports the functions required of it and remains responsive through the use of threading techniques. By means of technical testing, an issue has been identified with the time taken to import log information from the XML files into the database. User feedback on the issue of the time taken to load log data into the database has shown a mixed response as to how long users would be prepared to wait. It has been found that all those who took part in the user testing preferred the prototype for getting an overview of log data and the majority preferred it for filtering log records.

The following section contains an evaluation of the technical and user feedback and how the results from this section relate to the aim of improving upon current methods of collection, correlating and presenting event information in the form of a timeline.

6 Evaluation

6.1 Introduction

This evaluation will cover the solution produced, from both a technical (through discussion of the test results) and non technical (analysis of the user feedback) perspective. Through analysis of the test results and user feedback it will be seen how the prototype system performs, culminating in the opinions of a security professional, Sonya Buczyn, IT Security Officer at East Lothian Council. There will also be discussion of how the project was managed.

6.2 Evaluation of technical implementation and user feedback

6.2.1 Functionality and Usability

Going back to the conclusions of the literature review and PACT analysis carried out at the beginning of the design stage, it was determined that the system would be targeted at those with a good knowledge of computing. The system would need to provide a means of gathering, filtering and displaying log data in the form of a timeline.

These basic features were implemented as intended. The user can gather in log data from the Windows event logs and log process and file activity on a target machine. This log data can then be imported into the central storage facility (database) and then be filtered to output the detailed information required and charted in the form of a timeline.

The results from the user feedback clearly indicate that the prototype system successfully achieved the aim of improving upon existing methods from overview and filtering perspectives. By following principles of overview, filter and zoom with advanced querying functions such as AND OR as suggested by Shniederman (1996), it has been found that users prefer the prototype for an overview and filtering of data. From the comments made by users, the prototype in its current format was not preferred over the Microsoft Windows Event Viewer for viewing detailed log results. Had the prototype system included the ability for users to view more detailed information about log entry, such as Process ID (PID), and enabled the user to view the record individually rather than in a data grid which requires horizontal scrolling, the system would have performed better.

Whilst it was generally found that introducing colour to charts to highlight high and low points was found to be beneficial, as suggested by Stone (2006), an unexpected difficulty was found. The use of a traffic light style colour scheme of red, amber and green meant that the user with colour blindness found distinguishing between the red and green colours quite difficult. In Figure 20 the chart on the left is an example chart from the prototype. The image on the right simulates how the chart would appear to someone with deuteranopia, a form of colour blindness (Encyclopaedia Britannica, 2009). You will see that the difference between the colours of high and low values is noticeably reduced.

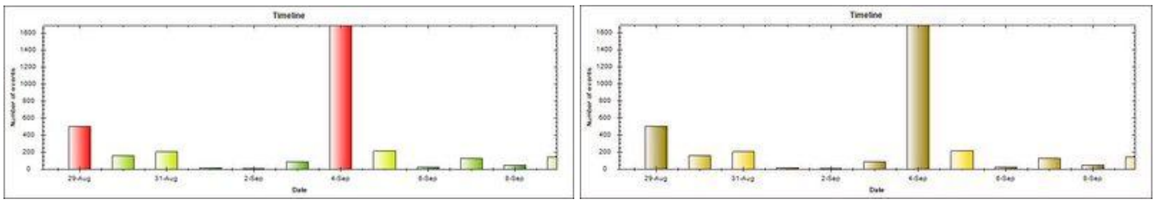


Figure 20: Problems with Red/Green colour scheme (Vischeck, 2009)

A future release of the prototype would have to take this into consideration and change, or at least provide the user with an option to change, the colours or tones used in order to ensure it could better be operated by those with this colour blindness condition.

The time-lining charts could be improved upon by utilising a charting system which would not only enable the user to zoom, as in the prototype, but also adjust the time resolution which was an issue identified during the literature review. In its current state, the prototype is inflexible in that the number of events recorded is grouped by day and cannot be changed.

An additional function which would have to be implemented in a future version of the solution would be checking of the SHA-256 hash which is recorded with each log entry in the XML files. As it stands, the prototype event reader, process logger and file watcher console applications create the SHA-256 hash. However to complete this feature, the hash would have to be tested when the data is read into the database. The design has provision for this feature so it could be added in a future release. In addition to using hash functions on the log data, the use of encryption would further enhance the data logging and store aspects of the system, ensuring that the log data held could not be accessed by unauthorised individuals although such a feature would almost certainly come at the expense of system performance as encryption would increase the processing overhead.

6.2.2 Performance and Scalability

The scalability of any log storage and filtering system is clearly important. In a criminal investigation, log information could be gathered from thousands of computer systems in order to build a case. From the results of the technical tests in Section 3.6.4 it is clear that as far as filtering the log information is concerned, the prototype system can remain stable and produce results in an acceptable time frame when viewing data over a period of months.

From the perspective of loading in the log data from XML files, the prototype does not prove to be scalable. In Section 3.6.3 it has been shown through testing that the performance of the system when loading in a relatively small log (up to 2MB) is very good, however as the log size increases the performance suffers considerably – thus proving that the current method is not sufficiently scalable. It is clear that loading in logs is a major bottleneck in the system. Interestingly from the user feedback, as discussed in Section 3.6.7, it can be concluded opinion is divided upon users as to how long is an acceptable length of time for the system to import the log data into the database. However it is clear that there would become a point where the system in its current state would become unusable due to the length of time taken to load in logs.

A possible solution to the problem of the time taken to load in the logs could be to update the database automatically whenever a new event occurs. In the case of a workplace where the storage of log information could be centralised it would be possible to develop further the logging applications, which run on the client machines, in order to have them automatically send log entries to the central database as each event occurs. This would negate the need to load in vast quantities of log data into the database whenever an investigation took place. However, this method could not be relied upon due to the possibility of an attacker disconnecting the equipment from the network, or indeed would not be of use to an outside agency (such as the police) that may be gathering data from machines across networks.

6.2.3 Robustness

From the results of Experiments 1 and 4 it is clear that the prototype system does not perform as intended under certain situations. This can be excused to a degree given that the solution is intended as a prototype to demonstrate concepts and is not intended to be ready for use within a normal working environment. From the results of Experiment 1, Section 5.1.2, it can clearly be seen how the use of multithreading techniques is essential in order to that the program appears active to the user. From Experiment 4, see Section 5.1.5, it can be seen where further development would be necessary in order for the viewer application to handle situations where a user mistakenly loads in the same log data twice or closes the application whilst importing log data. These issues could be overcome by additional error checking and preventing the application being closed whilst data processing. In other respects the robustness of the solution appears to be satisfactory, handling unusual user interactions correctly and preserving the data held within the database.

6.3 Professional Review

In order to determine how the prototype system compared to existing systems in use within an IT Security environment and where such a system would fit into a security professional toolkit, the opinions of Sonya Buczyn, IT Security Officer at East Lothian Council were sought. Her role involves giving guidance to users on various aspects of IT security, developing policy for protecting systems and investigation in the event of an attack.

As part of the session with Mrs Buczyn she completed the same questionnaire as other users taking part in the user testing of the prototype. This led to an interesting answer to one of the questions. When answering the question as to how long the user would wait for log information to be imported into the database Mrs Buczyn agreed that speed was not too important, given that a large amount of information was involved and that the other tools she used in her investigations also took time to gather in event data. This answer fits in with the expectation, previously explored, that users with experience of event logging systems are more likely to be prepared to wait in excess of ten minutes for the system to import log information. In addition to the question on time, questions were asked as to how a system, such as the prototype, could fit into an investigation. The view of Mrs Buczyn was that the prototype could certainly be of use as an addition to the use of EnCase:

> "As you know we use EnCase here for any internal investigations which can be pretty complex and take a long time to search for evidence. I can certainly see a place for using your application alongside encase as this would give a really quick method for searching for specific events. Your app would be a great addition to my investigation 'toolkit'". (Buczyn, Appendix 2.3, 2009).

In addition it was stated that the prototype's design meant that the system was easy to use and the log information presented was "very clear".

As part of the discussion with Mrs Buczyn she revealed that with upcoming codes of conduct for UK councils the subject of event logging was an active topic. As such she had been looking at commercial event logging solutions for managing event information and was impressed with the prototype system:

> "I've seen a couple of logging applications recently and based on what I saw of your app today, yours compares favourably with the commercial ones for the actual core logging and searching process" (Buczyn, Appendix 2.3, 2009).

6.4 Critical analysis of work carried out

Looking back at how the project has been managed and the prototype implemented and tested it is possible to identify a number of areas where work could have been carried out in a more efficient and robust manner.

6.4.1 Evaluation of implementation and testing

Several issues have been identified in relation to how the prototype system was implemented and tested. Although an overview design was formulated the technical implementation was not sufficiently planned out before coding began. This resulted in more time being spent on the implementation than might have been required as potential issues could have been identified before commencing the application development. It would have been better to document the program structure by means of Unified Modelling Language (UML) diagrams in advance as this would have both served as a template for the construction of the prototype and served as documentation of the implementation for future reference. The use of WPF and the .NET framework enabled rapid development of the prototype for Microsoft Windows Platforms. Unfortunately, these programming technologies do not prove to be compatible with other platforms, especially Unix which is used to a large extent in industry.

Testing was separated into technical and user testing. This worked well and produced interesting results, however improvements could be made. In order to improve the accuracy of the technical testing a virtual machine could have been used. This would have allowed for an identical test bed for each experiment, resulting in test results which are less prone to interference from other processes which may be running on a machine that is not dedicated for testing. In addition, technical testing was limited to three runs on the same hardware, for each experiment. A stronger set of results could have been achieved by testing the prototype on a range of hardware and software platforms and increasing the number of test runs. For example, if each test was run 10 times and the highest and lowest values excluded, then the median result taken, this would likely provide a more accurate result and further mitigate against any external influences.

The user testing could have been conducted on a larger and more wide ranging set of users. This may have provided a more conclusive set of results for the questions which, based on current results, had a mixed set of answers. Users were not specifically asked how they rated their experience of using event logging systems. Had this been asked then it would have been possible to correlate experience against the results, particularly for questions asking users to indicate their preferred system.

6.4.2 Project management

Appendix 1.3 contains the project plan which was devised ahead of beginning work on the project. In general the project progressed according to the schedule, however with the benefit of hindsight it would have been beneficial to allow more time for the testing and evaluation as this part of the project overran into the time allocated for checking the report over and making adjustments.

7 Conclusions and Future Work

7.1 Introduction

This chapter reviews the main findings of the project. The main findings from the technical and user testing are discussed in relation to discoveries made from carrying out the literature review. The chapter concludes by identifying areas of further research and development into gathering, storing and displaying log information in the form of a timeline.

7.2 Conclusions

The aim of this project was to explore and improve upon the methods of recording and displaying log information in such a way that timelines of events could be established. In light of the findings of the work of others, a prototype system was developed which incorporated the ideas and principles gained from a number of sources in the field of digital forensics, log management and visualisation.

The prototype system was designed and developed in a rapid prototyping fashion. This made it possible to design, implement and evaluate means of improving upon existing event recording and presentation methods within the limited time scale of an honours project. A number of technologies were used in order to realise the goal of developing the prototype. Through use of WPF and the .NET framework it was possible to construct a prototype which can operate on the latest Microsoft Windows platforms, such as Windows Vista and Windows 7.

Through means of a literature review it was found that visualisation could greatly enhance a user's ability to spot abnormalities in log data and make sense of large quantities of data. In a previous study by Shahar et al. (2005), it was found that visualisation allowed users to perform tasks more quickly than using traditional methods alone. Through means of user feedback, a similar result has been found, with users preferring the prototype solution to the Windows Event Viewer and text editor methods for viewing log information from overview and filtering perspectives.

The major issue of log correlation and management has been partially overcome by using a consistent event record format and storing the log information in an SQL database. Through testing it has been found that this was a good solution for the need for storage and filtering options. However, prior to being stored in the database, the design of the prototype system as a whole, involves storing individual logs in a custom XML format. Whilst the idea of storing logs in an intermediary XML format was brought about following research into the problems of proprietary formats, it was found during testing that importing the information in these XML files into the database was a bottleneck in a system which otherwise gave good performance.

During the implementation of the prototype, an additional feature of using a traffic light style colouring scheme, as suggested by the research in Section 2.5, was added to the system. This extra feature proved very useful to the majority of people who gave feedback on the system. However the introduction of colour introduced a new issue of accessibility where by the traffic light colour scheme proved to be difficult to work with for those who are colour blind. This could be addressed by altering the colours used when charting log information.

7.3 Future work

Much of the focus of this project has been around visualising log data but, due to the limitations of time, experience and graphing libraries available, it was not possible to take the charts further than representing totals over a period of days. Future work could involve enabling the user to zoom in on particular moments in time and have the charting system automatically adjust the time resolution. This could be taken still further by producing charts which attempt to link events together.

It was discussed during the literature review that the issue of time stamping event records was very important. Given the nature of the project it was considered that the issue of recording time through NTP or through a central system was out with the scope of the prototype. However, the reliance on accurate time stamps is clearly a major issue in digital forensics, with time stamping being used to determine the order in which events happened. Future work could involve building in a central time stamping and log storage system into the applications developed for the project.

During the implementation of the prototype processes logger a compromise was made as to how the user account was identified and recorded. By only recording the user account name logged onto the target system, rather than the account name which caused the process event, there is potential that event information could be misleading and an intrusion would be unnoticed. If an alternative to WMI could be used for identifying the process owner, then the system could be improved upon.

Given the modular approach to design, the system could be expanded to support different operating systems. It is common knowledge that the technical infrastructure of most businesses is a heterogeneous mix of platforms. This project has implemented the ground work by separating the log storage and filtering from the event gathering systems. Furthermore, the use of XML to store log data in a consistent format would allow for event gathering applications to be written for other systems such as Unix, Linux or even for more specialist embedded systems.

8 References

Casey, D. (2008). Turning log files into a security asset. *Network Security, 2008*(4), 4-7. Retrieved November 10, 2009, from Science Direct database.

The CEE Board. (2008). *Common Event Expression*. Retrieved April 2, 2009, from The CEE website: http://cee.mitre.org/docs/Common_Event_Expression_White_Paper_June_2008.pdf.

Deuteranopia. (2009). In *Encyclopaedia Britannica*. Retrieved November 11, 2009, from Encyclopaedia Britannica Online: http://www.britannica.com/EBchecked/topic/159679/deuteranopia

Forte, D. (2004). The 'ART' of log correlation: part 1: Tools and techniques for correlating events and log files. *Computer Fraud & Security, 2004*(6), 7-11. Retrieved March 5, 2009, from Science Direct database.

Gorge, M. (2007). Making sense of log management for security purposes - an approach to best practice log collection, analysis and management. *Computer Fraud & Security, 2007*(5), 5-10. Retrieved March 5, 2009, from Science Direct database.

Guidance Software. (2008). *EnCase Forensic Features and Functionality*. Retrieved November 10, 2009, from the Guidance Software website: http://www.guidancesoftware.com/resources-brochures.htm

Home Office. (2003). *Retention of communications data under Part 11: Anti-Terrorism, Crime & Security Act 2001. Voluntary Code of Practice*. Retrieved November 10, 2009, from http://www.opsi.gov.uk/si/si2003/draft/5b.pdf

Livnat, Y., Agutter, J., Moon, S., Erbacher, R. & Foresti, S. (2005). A Visualization Paradigm for Network Intrusion Detection. *Proceedings of the 2005 IEEE*. Retrieved May 16, 2009, from http://www.scientificcommons.org/40527712.

Marcella, A., & Menendez, D. (2008). *Cyber Forensics: A Field Manual for Collecting, Examining and Preserving Evidence of Computer Crimes* (2nd ed.). Auerbach Publications.

Marty, R. (2008). *Applied Security Visualization*. Indiana: Pearson Education, Inc.

Microsoft. (2008). *XAML Overview*. Retrieved November 8, 2009, from http://msdn.microsoft.com/en-us/library/ms752059%28classic%29.aspx

Microsoft. (2009). *Naming conventions in Active Directory for computers, domains, sites, and OUs*. Retrieved November 15, 2009, from http://support.microsoft.com/kb/909264

National Institute of Justice. (2001). *The Electronic Crime Scene Investigation Guide: A Guide for First Responders*. Retrieved November 20, 2009, from http://www.ncjrs.gov/pdffiles1/nij/187736.pdf

National Institute of Standards and Technology. (2009). *Secure Hashing*. Retrieved November 10, 2009, from http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html

Pasquinucci, A. (2007). The difficult art of managing logs. *Computer Fraud & Security, 2007*(10), 5-7. Retrieved March 5, 2009, from Science Direct database.

Porter, D. (2003). Insider Fraud: Spotting The Wolf In Sheep's Clothing. *Computer Fraud & Security, 2003*(4), 12-15. Retrieved November 15, 2009, from Science Direct database.

Schuster, A. (2007). Introducing the Microsoft Vista event log file format. *Digital Investigation, 4*(1), 65-72. Retrieved June 4, 2009, from Science Direct database.

Shahar, Y., Goren-Bar, D., Boaz, D. & Tahan G. (2005). Distributed, intelligent, interactive visualization and exploration of time-oriented clinical data and their abstractions, *Artificial Intelligence in Medicine, 2006*(38), 115-135. Retrieved April 20, 2009, from Science Direct database.

Shneiderman, B. (1996). The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations, *Proc. 1996 IEEE Conference on Visual Languages,* 336-343. Retrieved May 30, 2009, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.48.9138&rep=rep1&type=pdf

Stone, M (2006). *Choosing Colors for Data Visualization*. Retrieved June 5, 2009, from http://www.perceptualedge.com/articles/b-eye/choosing_colors.pdf

Teerlink, S., & Erbacher, R. (2006). Improving the computer forensic analysis process through visualization. *Next-generation cyber forensics, 48*(2), 71-75. Retrieved March 12, 2009 from Communications of the ACM database.

UsabilityNet. (2006). Rapid Prototyping Methods. Retrieved November 20, 2009, from the UsabilityNet website: http://www.usabilitynet.org/tools/rapid.htm

US-CERT. (2008). Computer Forensics. Retrieved March 10, 2009, from US-CERT: Publications: http://www.us-cert.gov/reading_room/forensics.pdf

Vischeck, (2009), *VischeckImage*, Retrieved October 25, 2009, from The Vischeck website: http://vischeck.com/vischeck/vischeckImage.php

W3Schools, (2009). *XML Elements*. Retrieved June 4, 2009, from The W3Schools website: http://www.w3schools.com/xml/xml_elements.asp

ZedGraph. (2007). *ZedGraphWiki*. Retrieved May 8, 2009, from The ZedGraph website: http://www.zedgraph.org