Analysis and Evaluation of the Windows Event Log for Forensic Purposes

CO42019 Honour Project

UNDERGRADUATE PROJECT DISSERTATION

Submitted in partial fulfilment of the requirements of
Napier University for the degree of
Bachelor of Science with Honours in Networked Computing

Barrie Codona
06007743
BSc (Hons) Network Computing

Supervisor:
Prof. William Buchanan

Second Marker:
Dr. Gordon Russell

Authorship Declaration

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

1. Where I have consulted the published work of others this is always clearly attributed.
2. 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.
3. I have acknowledged all main sources of help.
4. 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.
5. I have read and understand the penalties associated with plagiarism.

Signed

Name: Barrie Codona

Matric No.: 06007743

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

The windows event log is used in digital forensic cases, but unfortunately it is flawed in many ways, and often cannot be seen as a verifiable method of determining events. In the past few years there have been a few highly publicised cases where the data that is contained within the event log is used to successfully secure a conviction. The aim of this dissertation is to develop a solution that addresses the flaws in the Windows event logging service. Through research carried out it had been found that it was possible to disable the event log service. This then allowed for important data to be modified, such as usernames, computer names, times and dates. It was also noted that an event log from one machine could successfully be transplanted into another without any problems. All of these vulnerabilities involved having access to, and being able to edit, the event log files.

Based upon the research done, an event logging application was developed using C# and the Microsoft .NET framework. It makes use of RSA and AES encryption and HMAC hash signatures to improve the integrity of the data. The application is divided up into three components, an event logger which monitors specific files and folders within a computer system and sends alerts to the data archiving system in an XML format, and an event viewer that presents the events in a readable format to the user.

The performances of the symmetric and asymmetric encryption were tested against each other. It had been found that the symmetric encryption was 800% faster than asymmetric encryption. Also the HMAC hash signatures were tested to see how long it would take to do a brute force attack on them. It was discovered that approximately 21,093 keys were processed every second, this was then compared to the key entropy and how a longer random key would be harder to break.

Table of Contents

Table of Figures

Acknowledgements

I would like to thank Professor William Buchanan, for his invaluable guidance and support throughout this project. In addition, I would like to thank Dr Gordon Russell for being part of the marking process.

1 Introduction

1.1 Project Overview

The windows event log is used in digital forensic cases, but unfortunately it is flawed in many ways, and often cannot be seen as a verifiable method of determining events. In the past few years there have been a few highly publicised cases where the data that is contained within the event log is used to successfully secure a conviction.

Dr Harold Shipman used a database program for storing information about his patients. Whenever anything was added or modified in the database it automatically filled in the time and date. Dr Shipman used his knowledge of computers to rollback the system time and date so that he could enter false information about his patients and their medical conditions after he had killed them. What Dr Shipman was unaware of was that when the system time and date is altered it shows up in the Windows event log. This information was successfully used in court to prove that he had falsified his patient's records after they had past away.

In America the collapse of Enron and WorldCom spurred the introduction of the Sarbanes-Oxley Act in 2002, this piece of US legislation places greater focus on the auditing process within an IT department. It expects that the IT staff will risk assess the auditing process and introduce controls that increase the security and integrity of this process. Although this piece of legislation only currently applies in America, the UK's Financial Services Authority is in the process of implementing a similar piece of legislation here in the UK. It is currently being referred to as Europe-SOX or SOX-Lite and, just like its American big brother, it too places greater emphasis on the credibility of audit logs.

As previously mentioned these logs can be used in digital forensic cases to provide an insight into what the computer system has been used for, when it has been used and who was using it. Unfortunately the audit logs are not stored securely on the computer system. Anyone that has administrative privileges can turn off the logging service. They can subsequently edit the contents of the logs, delete the logs and they can replace the entire log with one from another machine. Physical access to the computer

system would also provide an attacker with the opportunity of removing the hard disc drives and either manipulating the data using another machine or simply destroying the drives. All of these evidence tampering techniques would make the digital investigation much harder and a more time consuming process.

1.2 Background

From as far back as the 1970's computer professionals have identified the need for computer security in distributed systems that contain sensitive information; part of the security package that is built into operating systems is the ability to audit events that have occured on a computer system. For the Windows user, this consists of the Windows event log which is a customisable audit log. It was originally designed as a diagnostic tool to identify any problems with the operating system and it also allows the user some degree of control over what type of events are recorded. In a commercial environment the auditing will be defined as part of the company's security policy. This will involve recording of events, via the use of domain controllers, when users are successfully logging on and off, and which host machines they are using. Although the domain controllers are used to capture the event data it is also possible to setup and store the log files on a separate log server. This server could be protected from network based attacks by using a firewall which only allows traffic from the domain controllers, see Figure 1.

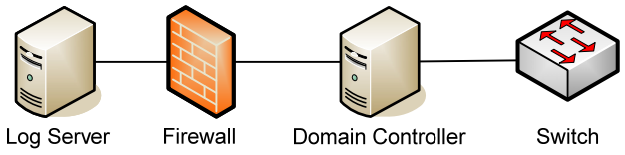


Figure 1: Log server protected by a firewall

However, as previously mentioned, there are a number of flaws with the Windows event logging service. These flaws not only relate to the security of the information after it has been written to disc, but also to what information it is actually capturing, storing and how it is stored.

1.3 Aims and Objectives

The aim of this project is:

1. To develop a solution that addresses the flaws in the Windows event logging service.

To achieve the aim, the following objectives of this project are to:

1. Investigate current research that is occurring in the field of Event Log Management and Digital Forensics.
2. Produce an analysis of the weaknesses in the Windows event log.
3. Design a suitable piece of software using an appropriate methodology.
4. Evaluate using a proven methodology.

1.4 Thesis Structure

Chapter 1 **Introduction.** This chapter will present some background information and define the aims and objectives for the project.

Chapter 2 **Literature Review.** This chapter will investigate the current research that is occurring in the field of Event Log Management and Digital Forensics.

Chapter 3 **Windows Event Log.** This chapter provides an investigation into the Windows Event logging service.

Chapter 4 **Design.** This chapter will provide an overview of the proposed software prototype that is to be developed.

Chapter 5 **Implementation.** This chapter will introduce the prototyped software that has been developed and a series of tests that will be designed to ensure that the software fulfils the requirements.

Chapter 6 **Evaluation.** This chapter presents the tested results of the prototyped system that has been previously developed.

Chapter 7 **Conclusions.** This chapter will provide a critical analysis of the project, a discussion into further work, and a description of the type of environment where the development of this project would be beneficial.

Chapter 8 **References.**

Chapter 9 **Appendices.**

2 Literature Review

2.1 Introduction

The aim of this literature review is to present to the reader current techniques and research that is being utilised and developed in the Log Management field, and how Digital Forensics are used to analyse these log files. It will also take a look at some of the techniques that are being used to disrupt a forensic investigation.

By identifying these techniques, it is intended that this will have an influence on the design of the system that is to be developed with the aim of improving the overall security and credibility of the Windows Event Logging system.

2.2 Digital Forensics

Crimes involving computers have increased dramatically over the last 10 years. This is due to the cost of owning a computer and having an internet connection decreasing, therefore more people are buying computers and more people are subsequently using the internet to facilitate their crimes. The Parliamentary Office of Science and Technology (Postnote, 2006) define some types of computer crime as storing illegal images, copyright violations, phishing, denial of service attacks and creating viruses. In similarity to the physical world, when a crime is committed there is always some form of evidence left behind, this is where digital forensics comes in.

Digital forensics is the science of being able to extract digital evidence from computers and other electronic devices “to aid the legal process” (Brown, 2006). According to Rogers (2005) of Purdue University, this is achieved by “identification, collection, examination and analysis” of the data that is contained on storage devices with the aim of discovering incriminating digital information (Nair, 2006).

This section presents to the reader some of the techniques that are currently used during such an investigation and how the Windows event log would be used to assist an investigation by recreating a timeline of events.

There are several stages that take place during an investigation and according to Palmer (2001) these are identification, preparation, approach strategy, preservation, collection, examination, analysis, presentation and returning evidence

Spafford (2004) describe this model as "The Abstract Digital Forensics Model". They also mention that this model is generally a good reflection of the forensic process, and that the four main stages are "preservation, collection, examination, and analysis".

- **Preservation.** This is the first phase of a digital investigation preserves the crime scene (Spafford, 2004). It involves creating a mirror copy of any hard discs and removable media, and if the computer is in a running state, creating a dump of its memory. This will allow the recreation of an exact copy of the system in a lab environment.
- **Collection.** This stage involves creating a record of the actual physical scene and creating a copy of any digital data that may be available. Cohen (2006) describes the following as good practice. To remove all equipment and cables, label, and record details, and search the area for diaries, notebooks and papers (especially look for passwords or other similar notes). Ask the user for passwords and record these, and then submit the equipment for forensic examination. In addition, The National Institute of Standards and Technology (NIST, 2006) state that the data should be captured using procedures that preserve the integrity of the data.
- **Examination.** This stage, according to Baryamureeba (2004) is an in-depth systematic search of evidence, the examination of collected data through a combination of automated and manual methods (NIST, 2006). But may also include other items such as log files, data files containing specific phrases, timestamps, and so on (Gladyshev, 2004).
- **Analysis.** The NIST (2006) describe this stage as analysing the results of the examination, using legally justifiable methods and techniques, it is also the determination of the significance, reconstructing fragments of data and drawing conclusions based on evidence found (Spafford, 2004).

The purpose of this is to obtain information that answers the questions that were raised during the identification stage and to ensure that the information that is collected will stand up in a court of law.

2.2.6 Forensic Techniques

To assist the Forensic Investigation a technique called data mining is used, this is the process of sorting through large volumes of data looking for particular patterns of information and complex relationships (Panigrahi, 2006) that might be of relevance. In particular Liu (2005) points out that if an investigator finds a suspicious file, they will search for other files with similar MAC attributes.

Another process in a forensic investigation is to search for deleted files and to try and recover them. This is made easier by the way in which NTFS handle files, it uses a relational database called the Master File Table (MFT) to store information about every file and its attributes on a system. The MFT reserves 1,024 bytes for each record that is contained within it, this is highlighted by:

> "One of the most interesting facts about the MFT is that it sometimes stores the actual file data along with all the system data relating to the file. Data stored inside the MFT is known as resident data". Armour Forensics (2005)

Armour Forensics also point out that when a file that has resident data is deleted and the MFT entry is overwritten by a file that is bigger than 1,024 bytes, then old file leaves behind some residual data.

2.3 Anti-Forensics

Since Digital Forensics is the science of retrieving digital evidence, then Digital Anti-Forensics is the science of thwarting such an investigation, by using tools and techniques to conceal or destroy information so that others cannot access it (Murphy, 2006). This section will look at some of the techniques that are used in anti forensics and they could be used against the Windows event log. One of the immediately noticeable effects of evidence tampering is a more difficult and time-consuming investigation (Forte 2007).

Ryan Harris of Purdue University (2006) proposes that anti-forensic attacks are grouped into the four categories, each of these are then subdivided into both physical and digital groups. The categories that he suggests are evidence destruction, evidence hiding, evidence source elimination and evidence counterfeiting.

- **Evidence Destruction.** Destroying evidence according to Harris (2006) partially or completely obliterates the evidence thus rendering it useless to a forensic investigation. One of the techniques that can be used to do this is called zero-footprinting. These are techniques that seek to eliminate all residual traces of an attack in order to prevent computer forensic operators from obtaining any results (Forte, 2007).
- **Evidence Hiding.** Ryan Harris (2006) states that hiding evidence is the act of removing evidence from view so that it is less likely to be incorporated into the forensic process. The evidence is not destroyed or manipulated however; it is just made less visible to the investigator. This can be achieved by using techniques like steganography or cryptography. Kessler (2004) describes steganography as the art of covered, or hidden, writing. Its purpose is hide information from a third party. Cryptography on the other hand is the art of secret writing; it differs from steganography because it does not hide the fact that a secret communication is taking place between two parties.
- **Evidence Source Elimination.** Disabling the Windows event log or any auditing system would help to hinder an investigation as data about the activities would never have be recorded; Harris (2006) writes “There is no need to destroy evidence since it is never created”. This could be similar to a burglar wearing gloves so that he does not leave any finger prints at the scene of the crime. This forward planning of shutting off an auditing system would imply that there had been some degree of planning involved.
- **Evidence Counterfeiting.** This involves changing the data that is contained within the event log, either the timestamps, usernames and/or machine names. At the kernel level it could also involve changing the Modified Accessed Created and Entry modified (MACE) attributes of a file. Evidence counterfeiting is the act of creating a faked version of the evidence which is designed to appear to be something else (Harris, 2006).

2.4 Log Management

The Windows event log contains lots of different information about events that have happened on the system. Microsoft TechNet (2007) describes the event log as a service that logs event messages issued by programs and the operating system. Event Log reports contain information that can be useful in diagnosing problems.

> "Because of the widespread deployment of networked servers, workstations, and other computing devices, and the ever-increasing number of threats against networks and systems, the number, volume, and variety of computer security logs has increased greatly. This has created the need for computer security log management, which is the process for generating, transmitting, storing, analyzing, and disposing of computer security log data." Souppaya (2006)

Souppaya (2006) continue to describe some of the uses of audit logs, and that they can provide information about such things as detecting attacks, fraud, and inappropriate usage. With all these events happening on a network it is important to have some sort of method for being able to ensure their integrity, keep them secure and monitor them.

2.4.1 Log Correlation

This is a technique that is used to bring together of all the different types of logs that are generated, and by using the timestamps on each of the events, create a timeline of these events.

A problem identified by Forte (2005) and Souppaya (2006) is the time stamping of logs, when logs are generated they are time stamped using the internal clock of the host and these can be inaccurate and this will have a knock on effect to the logs. For proper correlation the time stamping needs to be accurate, however Forte (2005) mentions that there is a reliance on NTP and this may open up a series of noted vulnerabilities. Souppaya (2006) gives an example of how this would cause a problem, "timestamps might indicate that Event A happened 45 seconds before Event B, when Event A actually happened two minutes after Event B".

Forte (2005) suggests a solution to this problem, see Figure 2; it involves the use of a single time stamping device that is attached to the network. Whenever an event is passed to the log repository it is stamped by the time stamping device, this dramatically increases the accuracy and credibility of the events. It also makes use of a Public Key Infrastructure (PKI) server. A Certificate Authority (CA) is a company that is used to hold a users public key, which is then used to authenticate them as being who they say they are. Here the PKI server authenticates the nodes so that fake events cannot be injected into the log repository.

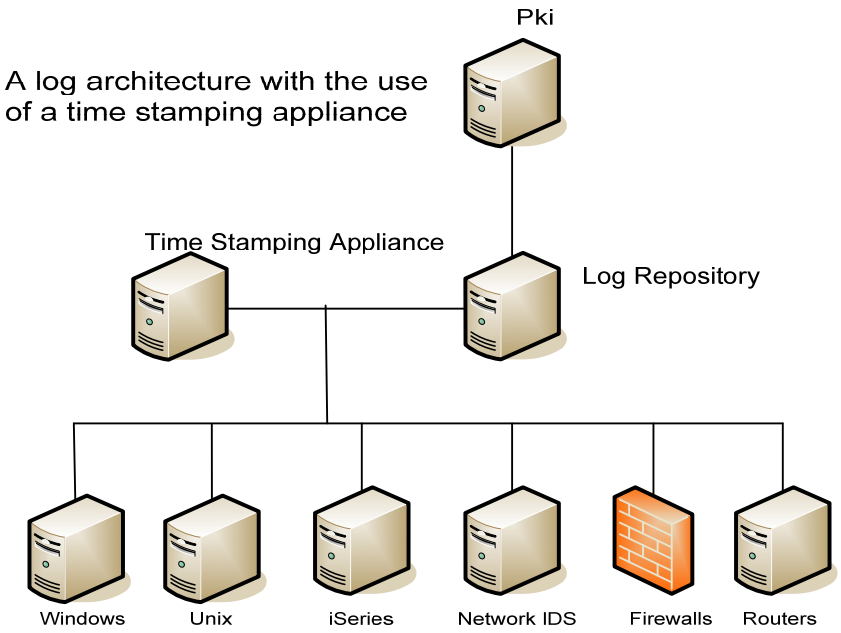


Figure 2: Log architecture with time stamping machine (Forte, 2005).

Forte (2005) does mention that although this would be easily implemented in an environment it would be expensive to do so. However this does present the idea of a central log server that would be used to store all the events from all the audit logs on a network.

A noted problem with this is that the log files use different formats (Souppaya, 2006) and as such need to be converted into a unified format. Some of the different formats are comma-separated or tab-separated text files, databases, syslog, simple setwork management protocol (SNMP), extensible markup language (XML), and binary files. Some logs are designed for humans to read, while others are not; some logs use standard formats, while others use proprietary formats (Souppaya, 2006).

A centralised log server brings with it many advantages, Westphal (2001) points out some of these advantages, centralised management of log files, maximised disk space usage, easier access for auditing purposes and a more secure method of retention. As well, adding encryption and check summing on top of a remote logging server adds yet another layer of security on these log files, which is always an advantage. The problem with this solution is that the data is venerable while it is travelling along the network to the log repository. Dario Forte, of Milan University, explains:

> "Log file integrity can be violated in several ways. An attacker might take advantage of a non-encrypted transmission channel between the acquisition and destination points to intercept and modify the transiting log. He might also spoof the IP sending the logs, making the log machine think it is receiving log entries and files that actually come from a different source." Forte (2005).

The general consensus is that a centralised log management system should be used to store, not only Windows event logs, but all the logs that are generated by all the devices on the network, and that this log server should be connected to an authentication server.

Alles (2004) have identified that a major problem is corporate fraud that is due to deliberate fraud between managers as this poses the greatest challenge to any audit system, there really is not any point in the best encryption and the best authentication servers if a corrupt manager can walk into the room where the logging server is being stored and gain access to the hard drive. So they recommend a black-box log file, something that is similar to a flight data recorder that you would find in an aeroplane and that the black box needs to be kept offsite and away from anyone that may want to gain access to it.

2.5 Conclusions

This chapter has presented some of the techniques that forensic investigators are currently using to reconstruct a timeline of events using event logs. It has also highlighted some of the methods that are used to defeat such a forensic investigation. Finally it has highlighted some of the ways in which audit logs are currently being

stored on servers, and some of the problems that can be associated with it, in particular the time-stamping.

Digital Forensic Investigators have the time consuming task of having to sift through Terabytes of data, looking for the smallest piece of information that could make or break a case. Unfortunately for them there is also a myriad of software out there that is designed to make their jobs harder, users can change the time stamps on file, and they can securely wipe data off their systems using software that makes several passes of writing random information where their file used to be. Using techniques like steganography illegal images can be hidden behind other seemingly acceptable pictures and files can also be encrypted on a system.

One thing that forensic investigators do have is the ability to create a timeline of events, to try and piece together what the accused person has been using a computer for. Sometimes this can be compromised too, the user may have just reinstalled the operating system, thus wiping most of the information from the hard drive, some data may still reside in the slack space that has not yet been written too, but there is the probability that the operating system has written over old event logs.

In a business environment logs are generally stored on a log server; this simplifies the task of having to review events that would have previously been located on several different devices. Although the log server may be locked away in a secure room somewhere in the building, someone will have access to that room, and they will be able to access the server and they will be able to modify or delete the logs that are stored on it, thus it would be better if they could be stored off-site.

It was pointed out that the time stamping of the events can be unreliable, so a central time stamping system needs to be used. It has also been suggested that the logs should be stored off site and by a company in a ‘write once read many’ environment and that they would not give physical access to the servers, and that the data on these servers contains a checksum to help validate the integrity of the logs and finally the logs should also be encrypted before they are sent to the log file storage company.

3 Evaluation of the Windows Event Log

3.1 Introduction

One of the objectives of this project is to "Produce an analysis of the weaknesses in the Windows event log." This chapter provides an investigation into the weaknesses of the Windows Event logging service; experiments are carried out that highlight just how severe some of these weaknesses are. Usernames, computer names, times and dates can all be changed, thus, concealing whatever information is desired.

By identifying these weaknesses at this stage, it will allow for a system to be developed that directly addresses these problems, and as such should help to provide a more authenticated event logging mechanism.

To carry out these tests virtualisation was used to provide a manageable test environment. VMWare (http://www.vmware.com) with a total of four virtual machines was used, two of them were Windows 2003 Servers and two of them were Windows XP Clients. Server A and Server B were both configured as Domain Controllers, DNS Servers and File Servers within the same domain. Clients A and B were both joined to the domain. Two user accounts were created User A and User B; these will be the accounts that are to be used in the testing. The two virtual servers had a Hex Editor installed on them, this would allow for the viewing and possible manipulation of the Event Log files.

3.2 Experiments

The following experiments range from being fairly simple tasks to moderately complex, the purpose of these experiments to be able to identify he weaknesses of the Event Log. These are:

Experiment 1 - Stopping and Restarting the Event Log. This experiment will show how it is possible for a user with administrative privileges to be able to stop and start the event logging service; although this may seem basic the technique involved is of fundamental importance to the rest of the experiments as the event log file is read only when the service is running.

Experiment 2 - Copying the Event Log. This experiment will demonstrate how it is possible to be able to take a copy of the Event Log while it is still running. This shows how it is possible to generate an event log file that can then either be edited or copied across to another machine.

Experiment 3 - Swapping the Event Log from one computer to another. This experiment will investigate if it is possible to be able to replace the Event Log on Server A with that of Server B, and if it is possible does it produce any errors. This i

Experiment 4 - Modify the 'logon ID'. This experiment will involve using a Hex Editor to modify the binary data that is contained within the Event Log, specifically the Logon ID that represents a specific user, for example try to replace User A with User B. This could be used by someone who intends on framing another person or covering their tracks.

Experiment 5 - How the event log stores the time. This experiment will take a closer look at how and where the Event Log stores the time that an event happened at. It will also analyse the format that is used. This leads onto the next experiment.

Experiment 6 - Modify the 32-bit time. This experiment will attempt to change the timestamp, using a Hex Editor, which has been applied to a specific event. If it is possible then this could be used to disrupt the timeline of events that have happened and as such would slow down a digital forensic investigation.

Experiment 7 - Automatically replacing the 'SecEvent.Evt' file. This experiment will look at the possibility of writing a script that could be used to automatically replace the Event Log, the purpose of this to be able to prove that it is possible automate this process and to identify what happens on the system.

Experiment 8 - Master File Table. This section will take a closer look at how NTFS stores files on a computer system. With this information an attempt will be made to try and modify the event log by directly accessing its data on the hard disk while the Event Logging Service is still running. The purpose of this is to be able to by-pass the need to have to top the event log service.

3.3 Experiment Results

A summary of the results of the above tests are contained within this section. The actual test data can be found in Appendix I.

Experiment 1 - Stopping and Restarting the Event Log. It was found that stopping and restarting the event logging service was a fairly simple task for a user with administrative privileges, there are two possible techniques both of which provide the same outcome. The first and easiest way is to navigate to Services Management Console which is located within the administrative tools folder of the control panel. Double click Event Log from the list of available services and set the 'startup type' to disabled, then restart the server. Another method is to adjust the startup value setting in the registry, namely 'HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet \Services\Eventlog' from 0x2 to 0x4. As with the previous method, this will not take effect until the server has been restarted.

When the server has restarted, an error will be displayed informing the user that 'at least one service or driver failed during system startup'. This is normal and will only be displayed once at the logon screen while the service is disabled. The process of restarting the service through the Service Management Console is merely a matter of changing the 'startup type' back to automatic, clicking apply and then clicking start. The service will restart without having to restart the server. It can also be restarted by changing the startup value in the registry back to 0x2, and to save having to reboot the machine it can then be started by typing the following in a command prompt window "net start eventlog".

Experiment 2 - Copying the Event Log. To simply make a copy of the Security Log, navigate to 'C:\Windows\System 32\Config\' and then right click and select copy on 'SecEvent.Evt', this can then simply be pasted into the same directory. It will then create an exact duplicate of the Security Event file called 'Copy of SecEvent.Evt'. This can be done while the event logging service is still running. Figure 3 shows this.

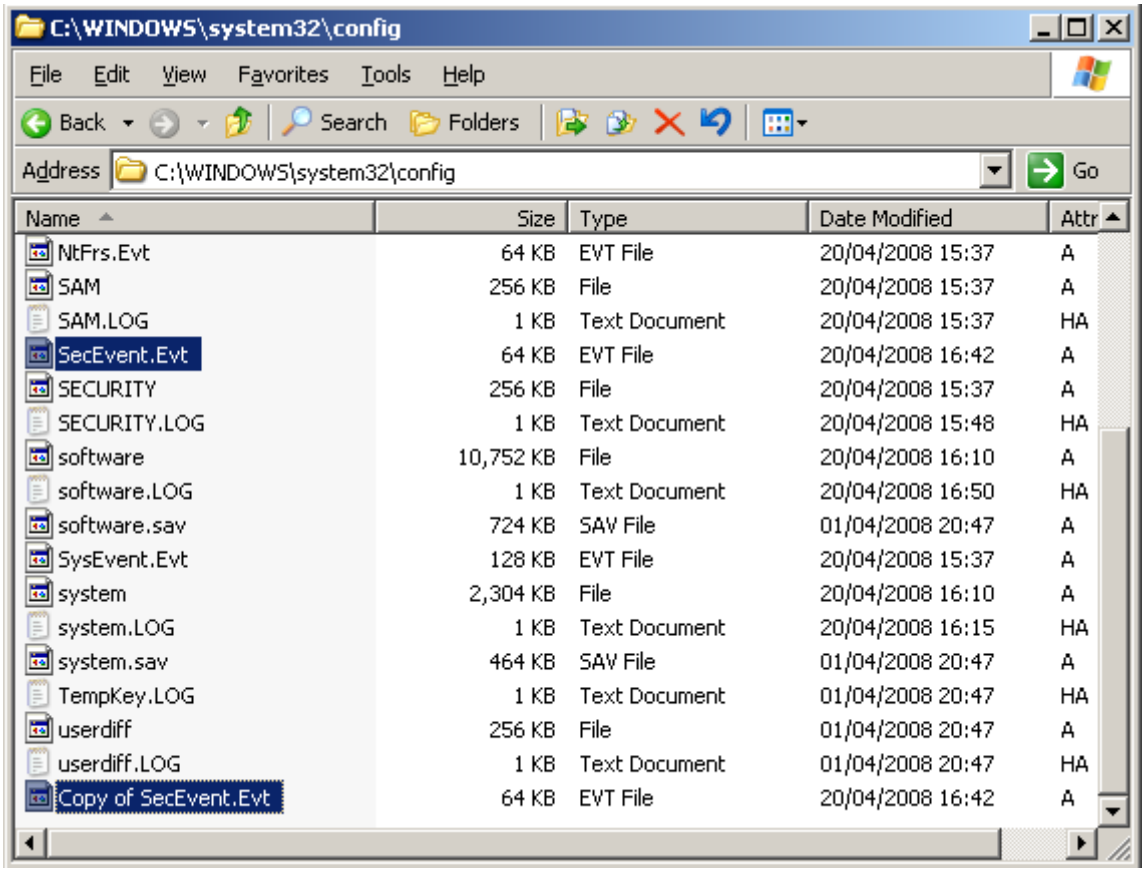


Figure 3: Copying the Windows security log.

With the results of this experiment combined with the results of previous one, it is now possible to attempt to replace the event log from one machine with that of another.

Experiment 3 - Swapping the Event Log from one computer to another. For this experiment a combination of the first two experiments was used, the Event Logging Service on Server B was set to disabled and the machine was restarted. The event log on Server B was then replaced with the one that had been copied from Server A and the event logging service restarted. Figure 4 shows this.

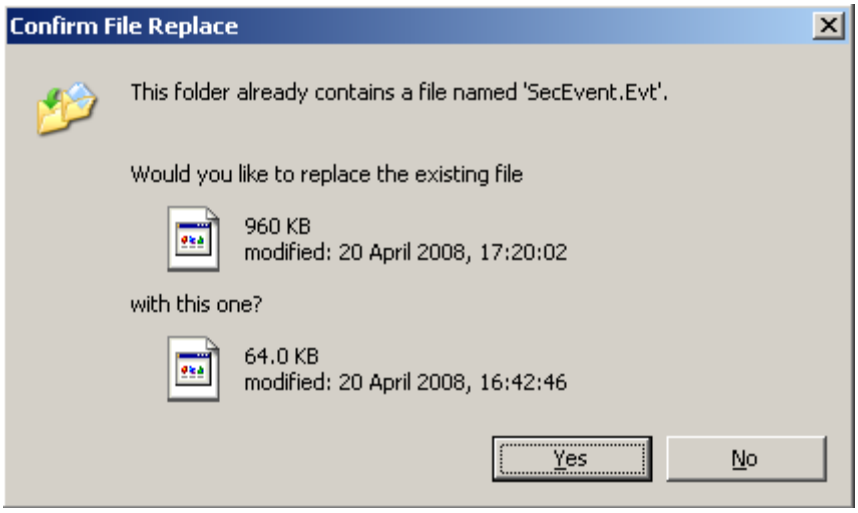


Figure 4: Replacing the Windows security log.

Server B successfully managed to accept the transplanted Security Event Log from Server A without producing any errors. This shows that it is possible to conceal events that have happened by simply copying a file to replace the original.

Experiment 4 - Modify the 'logon ID'. This experiment used a Hex Editor called 'Hex Workshop' called (http://www.hexworkshop.com) to view and modify the contents of the Security Log file. The data contained within the log file was in a fairly readable format, it is 8 bit ASCII that has been padded out to 16 bit. Thus the word 'Hello' is represented in the file as 'H.e.l.l.o.'.

First to be modified was the username, this was changed to a different one with the same number of characters in it, and then the computer name was modified, once again keeping the number of characters the same. When the event service was restarted it successfully showed that the both the username and computer name had successfully been modified.

This is highly significant as it shows that data contained within the event log can be modified and these changes will be accepted by the system. Furthermore, it also shows that it is possible for a perpetrator to either conceal their tracks or frame someone else.

Experiment 5 - How the event log stores the time. This experiment involved the use of a Hex Editor to analyse the content of the Security Event Log. Trial and error was used to locate the bits that held the date and time stamp, the Hex Editor used has a

'Data Inspector' window which translates the currently selected hexadecimal value into various different 16-bit, 32-bit and 64-bit values. Importantly it displays the value in 32-bit time.

An event was selected from the Event Viewer and a note of the time and date was taken, then using the Hex Editor, the file was manually searched until the value was found within the file. The value was reset to '0000 0000', this was then interpreted as being '00:00:00 01/01/1970', so from this the original hexadecimal value is taken to be the number of seconds that have elapsed since 1st January 1970, which is also known as "Unix Time" (Wikipedia, 2008).

There are a total of 64-bits that are used for the timestamp; these are two groupings of 32-bits both with the exact same value. For example '18:47:24 01/11/2007' is represented in hexadecimal as '3C1F 2A47', Figure 5 shows that in the event log this is stored as '3C1F 2A47 3C1F 2A47', note that it's the same value repeated.

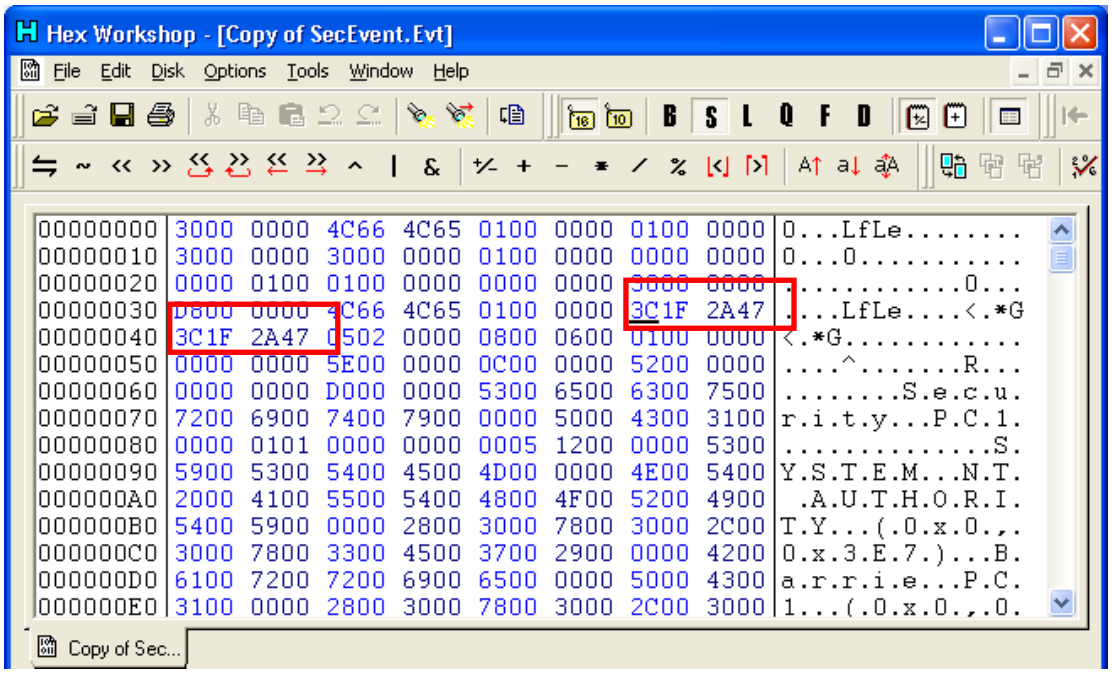


Figure 5: 32-bit time

Experiment 6 - Modify the 32-bit time. This experiment was a continuation of the last one; essentially it was done to see if the Event Log would produce any errors if a timestamp in a file had been modified. It had been noted that to adjust the time by 1 second, the left most bit needed to be incremented by one. For example '3C1F 2A47' is equal to '18:47:24 01/11/2007' and to increase this by 1 second the new hexadecimal value would have to be '3D1F 2A47'. When this was applied to the

Security Log and then restarted the Event Viewer did in fact display the time as expected without any errors. Figure 6 shows the first 32-bits being reset to '00000000'. Once again this experiment shows that the system is content working with event logs that have been modified.

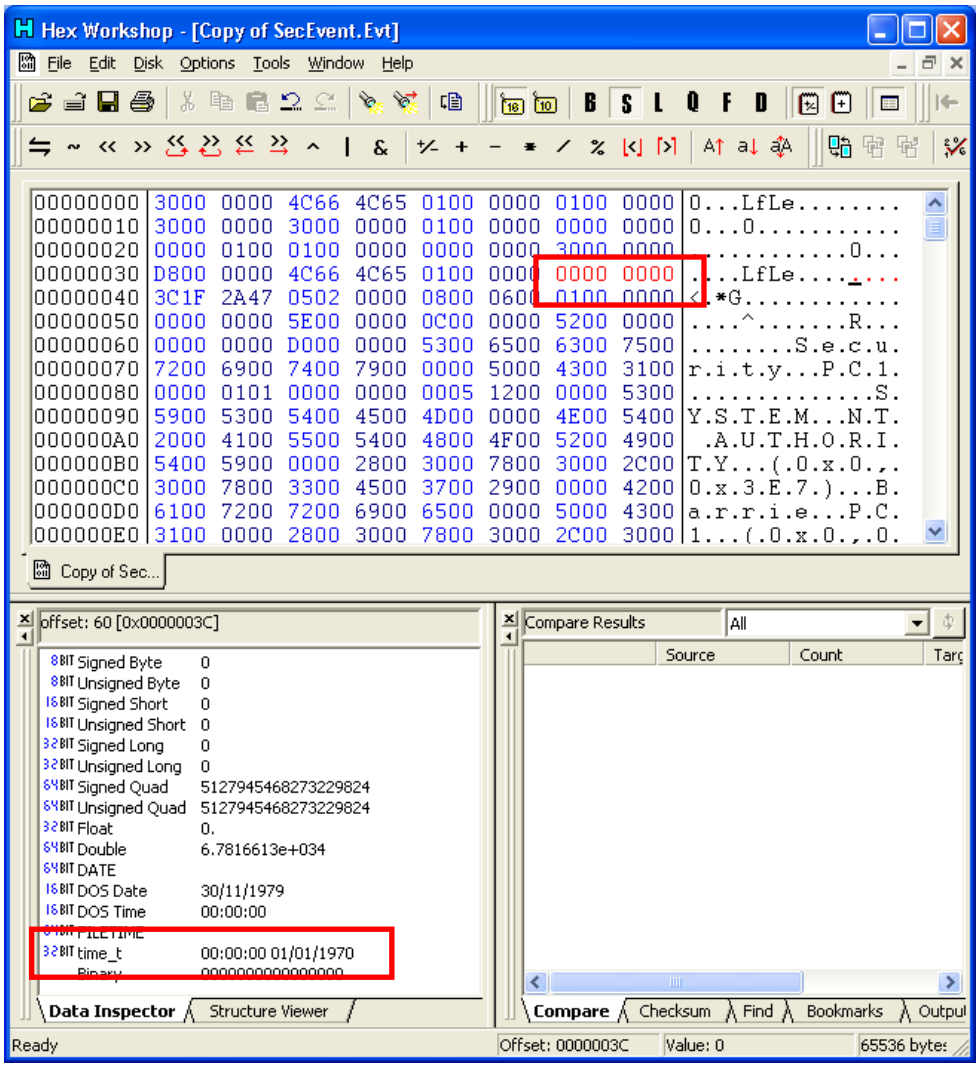


Figure 6: Modifying the 32-bit time.

Experiment 7 – Automatically replacing the 'SecEvent.Evt' file. For this experiment a console application was developed that would use the second technique for stopping and restarting the Event Logging Service. When the program is run checks the state of the 'startup' registry key, if the value is 0x2 it sets it to 0x4, it then creates an entry pointing to itself in the run once registry key, then it restarts the computer. When the machine restarts the program is automatically run, once again it checks the value of the event log 'startup' registry key, it should still be set to 0x4 and

in which case the program automatically replaces the Security Event Log with a predetermined file and then restarts the service.

Experiment 8 – Master File Table. For this experiment a program called Directory Snoop was used in conjunction with the Hex Editor. Directory Snoop (http://www.briggsoft.com/dsnoop.htm) is an application that allows for the inspection of the NTFS Master File Table. Which, as previously discussed in the Literature Review chapter, is a relation database that contains information about every file on the local disk? Directory Snoop was used to find the actual sectors of the local disk that was allocated to the Security Log. This sector was then opened up with the Hex Editor and some changes were made to the timestamp data. Upon viewing this information with the Directory Snoop it showed that the changes had actually been made, however when the Event Viewer was opened up it did not show these changes. This implied that that when the Event Log Service is first started it reads the contents of the event logs from disk into memory and does not read them again until it is restarted.

The server was then restarted to force the event log to reread the modified sectors, once again the Event Viewer was opened, however the changes had not taken effect. Directory Snoop was again used to check the sectors that had been modified and it showed that they had been restored to their original values. The Event Logging Service must have completely written the contents of its array in memory back to disk.

3.5 Conclusions

From the above experiments multiple vulnerabilities have been identified, it is possible to change the time that events happened at, usernames can be modified, the event log can be copied and pasted into another machine. The most surprising points learned was that there is no security in the event log at all, no hashing to see if it had been modified and nothing to stop an event log from one machine being used on another.

Being able to modify the event log only involved having access to the Security Event Log file and then modifying the data contained within. Thus, if the Event Log files

could be monitored for any copying, writing or deleting then it could be said that it would be possible to highlight a possible attack on the integrity of its data. Also, being able to monitor the status of the Event Logging Service, using the system registry, would highlight if it had been changed from automatic to disabled. The design of the system will be based upon these findings. It will disallow any physical access to the event logs and will also produce a hash signature that will highlight if any changes have been made.

4 Design

4.1 Introduction

One of the objectives of this project is to design a suitable software solution that is based upon research done. This chapter aims to deliver a prototype system that is based upon the findings in the previous Literature Review and Windows Event Log chapters. The main influence in the design of this system has to be that of the *black-box* concept that was previously discussed, in theory this would be a secure off-site data storage facility that would provide the ability to write custom event logs, without having access to be able to modify or delete them, essentially a write once environment. Each event will also have a hash signature attached to, so that if anyone attempts to modify them this will be highlighted.

4.2 High-Level Design

This application will consist of three separate components, an event logger, a data archiving system and a separate event viewer. The event logger will reside on the computer system that is to be monitored. When it is first started it will prompt the user to enter a key that will be used for generating a hash signature. It will watch for changes that happen to the file system, files being created modified, deleted and renamed. When one of these events happens it will create a hash signature for it. It is then encrypted and sent to the data archiving system.

The data archiving system receives the data from the event logger and decrypts it. It then applies a time stamp to the data before it is written to an event log. The event log is then read by the event viewer, it displays all the information about the selected event to the user in a format that is easily readable. It also allows the user to enter the key that was used for generating the original hash signature. The event viewer will then use this key to generate a new signature based upon the contents of the log file and compare it to the signature that is held on file for this particular event. So long as the data within the log file has not been modified the signatures will match. This will authenticate the message as being genuine. Figure 7 shows the communications that take place between the event logger and data archiving systems.

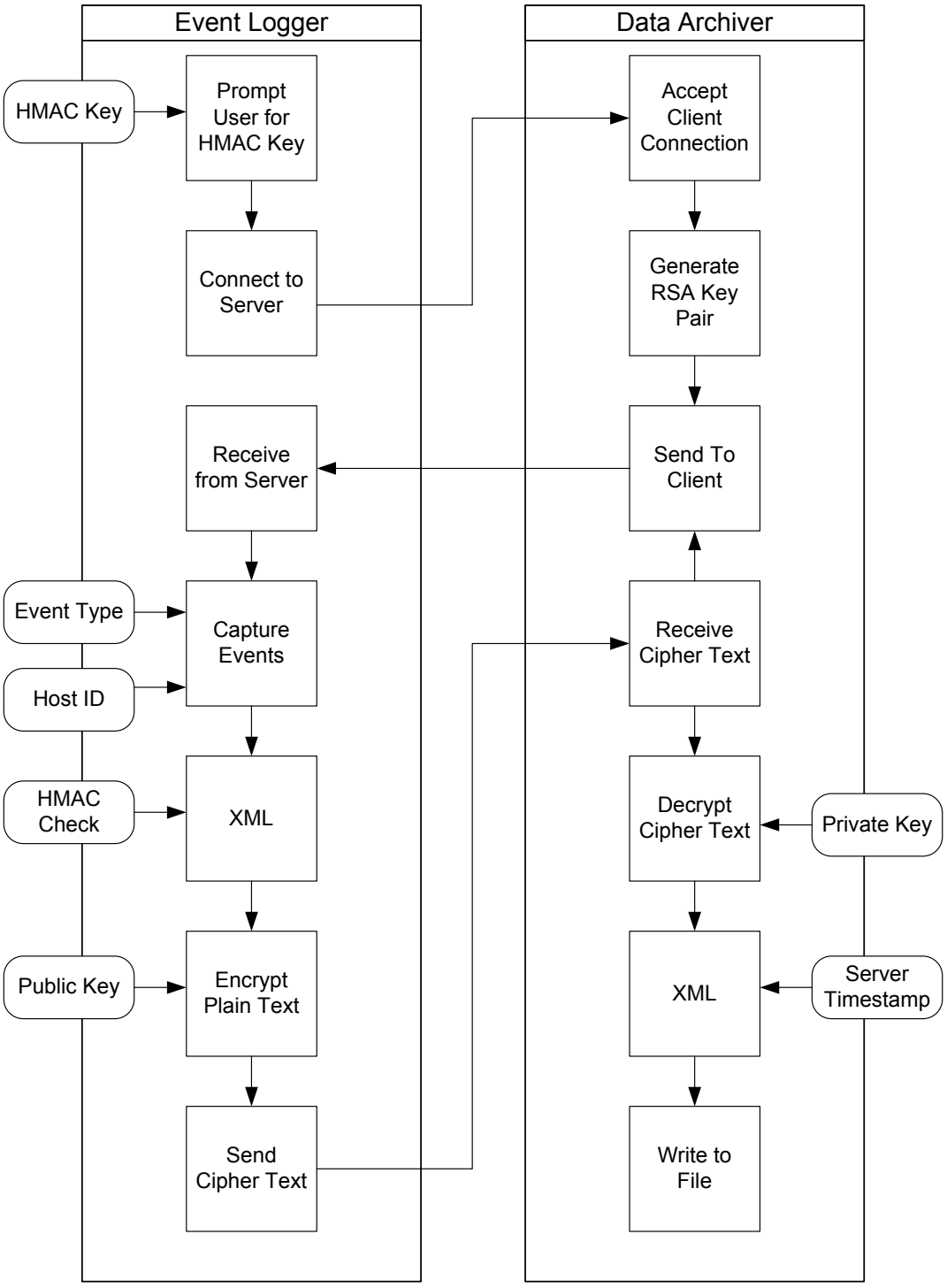


Figure 7: Event Logger Schematics

4.3 Event Logging

One flaw of the Windows Event Log is that it does not keep track of which users are accessing which data files and to see if they have been modified by that user. This

presents a problem that should also be addressed by the design of the software for this report. Instead of just monitoring the Event Log files, which are only accessed by the system when the server is started or stopped, it is proposed that the software being developed should be able to either monitor the entire system or specific directories.

The performance of the system could be compromised by an excessively large amount files to keep track of, however, the more data that is produced then a more accurate investigation can be performed.

The .NET Framework provides the 'FileSystemWatcher' class which is able to monitor if files have been accessed, modified, deleted or renamed, MSDN (2008). It can be set to monitor specific files within a specified directory or the entire system. Making this event driven will also make the program more efficient as far as processing is concerned.

As previously discussed in the Literature Review chapter, the formatting of the data that is contained within the log file is important. Firstly, for digital forensic purposes, the data that is captured and stored needs to be done in a manner that does not alter the original data. Secondly, the data should be in a format that is easily accessible to a variety of different applications.

Based upon these specifications the data that is being stored will be contained within the XML format. This essentially places every piece of information within its own opening and closing tags. For example, <name>Barrie</name>. The information that will be captured will consist of:

- The time and date that the event took place.
- The user that performed the action.
- What kind of action has been performed.
- And the path of the file.

4.4 Data Archiving System

To achieve the *black-box* concept, this will involve a client-server architecture, where the client is the machine that is capturing the events and the server or Data Archiving System will be the machine that is storing the log files.

Both will have to be able to pass a number of pieces of information between each other.

1. The Server is listening for a connection.
2. The Client connects to the Server.
3. The Server generates a random RSA key pair.
4. The Server sends its public key to the Client.
5. The Client receives the Servers public key and stores it in memory.
6. The Client sends events to the server.

Figure 8 shows the communication that takes place between the Client and Server. When the Server has received an encrypted event from the Client it will decrypt the message, and then add an additional timestamp, this will then be written to file.

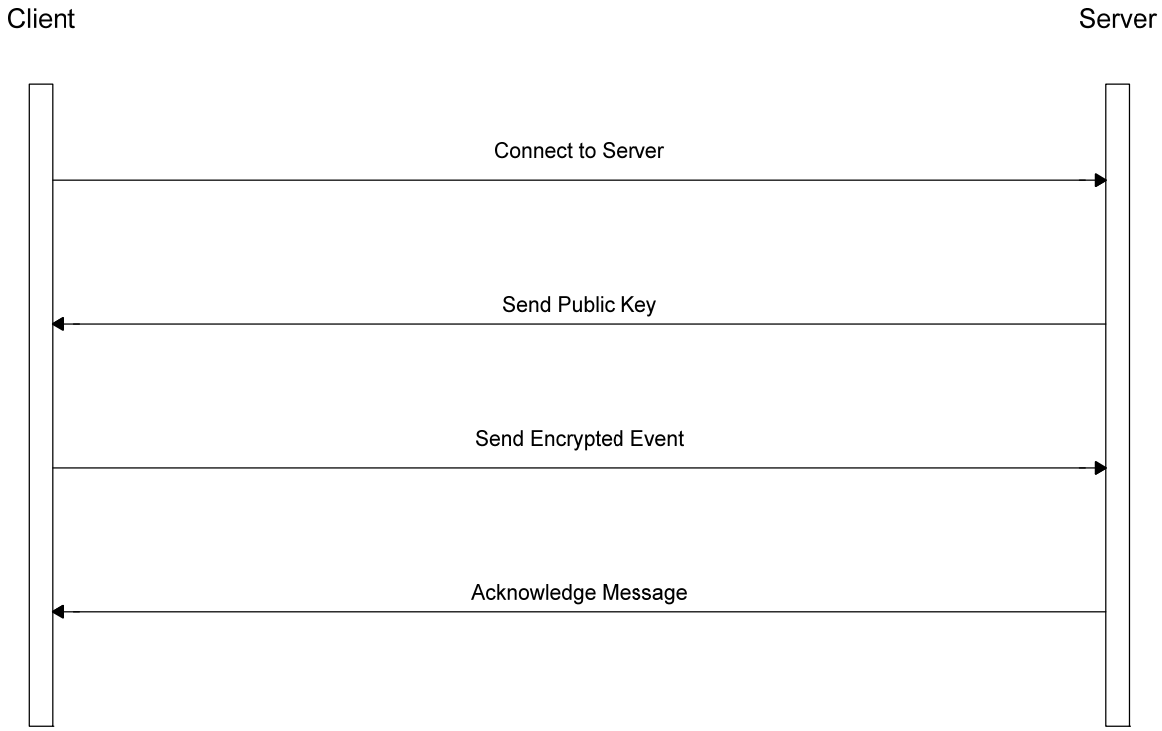


Figure 8: Client-Server prototype

4.5 Encryption and Authentication

There are two main types of encryption, symmetric and asymmetric, as a brief overview, asymmetric encryption uses two separate keys is more secure than symmetric, which only uses one key, but it is dramatically slower and is generally

only used for encrypting small amounts of information, this is due to the large calculations that are involved.

Two prototype systems will be developed, one that uses asymmetric encryption to secure all communications between the client and server. The other will use symmetric encryption to encrypt the bulk of the communications, but will transfer the shared key using asymmetric encryption, this system will be faster but it should be interesting to see if it makes any detrimental impact on the system.

To authenticate the messages HMAC will be used. RFC2104 (1997) describes HMAC as a mechanism for message authentication using cryptographic hash functions. When the original event is generated and had its timestamp and user tags added, it will then go through a process of having a unique HMAC checksum generated for each event. This will then be tagged onto the end of the event before it gets sent to the Server. Figure 9 shows this.

```
<clientTime>.....</clientTime>
<user>      .....</user>
<changeType>.....</changeType>
<fullPath>.....</fullPath>
```

Figure 9: Client generated XML string

The previous diagram illustrated the event information before it has been passed through the HMAC checking process, Figure 10 below shows the same event that has been checked and had its HMAC tags added.

```
<clientTime>.....</clientTime>
<user>      .....</user>
<changeType>.....</changeType>
<fullPath>.....</fullPath>
<hmacCheck>.....</hmacCheck>
```

Figure 10: Client XML string with HMAC

Figure 11 shows the event information that has been received by the Data Archiving System where it has also been time stamped. This is how it will look when it is stored in the event log on the Data Archiving System.

```
<serverTime>.....</serverTime>
<clientTime>.....</clientTime>
<user>          .....</user>
<changeType>.....</changeType>
<fullPath>.....</fullPath>
<hmacCheck>.....</hmacCheck>
```

Figure 11: Server XML string

4.5 Event Viewer

The Event Viewer Application will be used to view the contents of the event log that has been generated and stored on the Server. If will display the contents of each Event, for example the different Client and Server timestamps that have been generated. It will also display details about the event that have taken place, namely what files have been modified, changed, deleted or renamed. As well, it will also display the HMAC checksum that was originally generated and will also recalculate a new checksum value based solely upon the information in the Event Log. This will then be compared to the original. So long as the user inputs the same key that was originally used, this check will alert the user if the data has been changed. Figure 12 shows the graphical user interface that has been designed for the application.

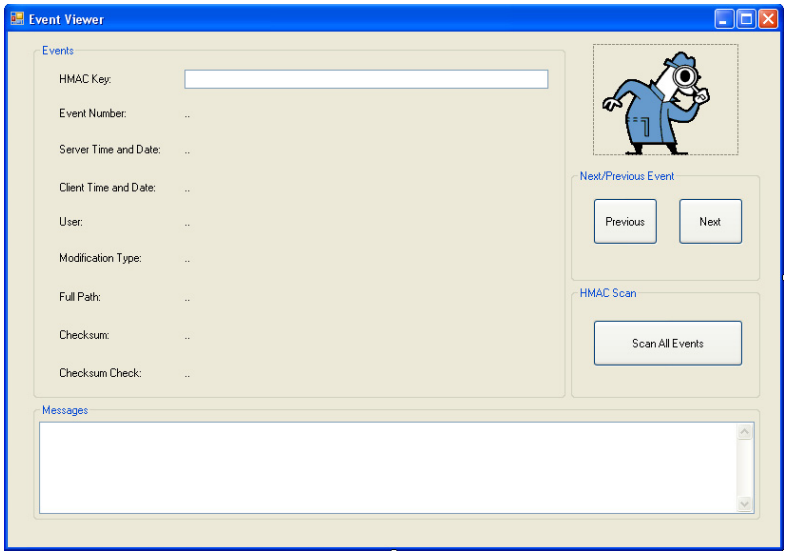


Figure 12: The event viewer interface

This will allow the user to be able to access and view each of the Events that's are stored in the Event Log, it will work by reading the Event Log file into an array and then allow the user to access each element of the array, as each 'record' is accessed the application will use the XML tags to sort the data into its correct field. The

application will also provide the ability to search through all the events looking or any that fail the checksum check.

4.6 Pseudo Code

Pseudo code is used in software development to give a description in a readable form of the basic functions of a program. The purpose of this is to make it more readable for humans, so they can understand what the program is doing, without overcomplicating the code and introducing complex algorithms.

4.6.1 Client Side Pseudo Code

Figure 13 is the pseudo code that the client application will use. As it can be seen the client attempts to connect to the Data Archiving System using port 13000. This port has been selected as it outside of the range of commonly used ports. After the connection has been established the client receives the public key from the Data Archiving System. This key is then used to encrypt all the data that is sent to the Data Archiving System.

```
Prompt user to enter a key for use with HMAC
Connect to the server on port 13000
Receive public key from server
Do Continuously
{
      Monitor for files being 'modified'
      If a file is 'modified'
      {
            Get system time
            Get username
            Get event type (changed/created/deleted/renamed)
            Get path of file
            Generate XML string
            Generate HMAC checksum
            Add HMAC to XML String
            Encrypt with Servers public key
            Send to Server
      }
}
```

Figure 13: Client side pseudo code

4.6.2 Data Archiving System Pseudo Code

The data Archiving System listens for a connection using port 13000, after a connection has been established it generates a random RSA key pair, one public and one private key. The public key is sent to the client machine and the private key is kept and used for decrypting all the data that is sent to it. The data received is then saved to disk. It continues this process until the client application disconnects, Figure 14 shows this.

```
Listen for connections on port 13000
If client connects
{
      Generate new RSA key pair
      Send public key to client
      While Client is still connected
      {
            Listen for messages
            Decrypt Message with private key
            Time stamp incoming message
            Write message to file
      }
}
```

Figure 14: Server side pseudo code

4.6.3 Event Viewer Pseudo Code

Figure 15 shows the pseudo code for the event viewer, initially it reads the contents of the event log into an array as a complete XML string. It then extracts the information from the XML string as it needs it. Based up the HMAC key that has been entered by the user, it also calculates what the HMAC value should be for that particular XML string and informs the user if it matches the HMAC signature that is contained within the XML string.

```
Read Event Log File into Array
Extract data from XML tags and populate fields
If user clicks ‘next’ or ‘previous’
{
      Display next or previous event accordingly
      Calculate checksum based on inputted key
}
```

Figure 15: Event viewer pseudo code

4.7 Testing Design

The system will be tested to ensure that its performance does not impede its ability to detect and log events, and that the accuracy of the data being collected is correct, and that the data being captured, transmitted and stored is being done so in a secure and authenticated manner.

The previous tests that were done with the Windows Event Log will be rerun with the new application running, the purpose is to see if it can highlight if the Event Log has been modified. As well as this the application will be stress tested to see what its limits are. Figure 16 shows the tests that will be carried out.

Testing	
Type	**Description**
Performance	How long to encrypt/decrypt?
	How much space will the log use?
Accuracy	How many events will it successfully capture?
	Is there a limit on the size of the directory path?
Security	Does HMAC work?
	Can HMAC withstand a brute force attack?

Figure 16: Testing design

4.8 Test Suite Design

An application that will automate part of the testing process, using scripts, has been developed. Its primary objective is to generate a substantial number of events that will be captured by the event logging application. This will allow the capabilities of the application to be measured and depending upon the results changes can be made to suite.

As previously mentioned the .NET FileSystemWatcher class is used to capture events that are happening to files within specified directories. The testing application will allow for the creation, modification, renaming and deleting of a variable number of files within a predetermined directory. An option of the testing application will be to automatically modify some of the Windows Event Log files while it is also generating large amounts of events.

Figure 17 shows the pseudo code of the process that the application will go through for generating a large numbers of files. Using a For Loop it will continue to create files a predetermined number of times. Each of these files will also be uniquely numbered in their filename.

```
Repeat the following 'x' number of times
{
      Create a new file called 'Sample'x'.txt' in the test
      directory
}
```

Figure 17: Generate a large number of files pseudo code

To provide some degree of error checking this can be slightly modified to perform a check to see if the file already exists. Figure 18 shows how the application will use the unique filename numbering to check if the file exists. This technique will also be used for automatically deleting and modifying the files.

```
Repeat the following 'x' number of times
{
      If file 'Sample'x'.txt does not exist
      {
            Create a new file called 'Sample'x'.txt' in the test
            directory
      }
}
```

Figure 18: Generate files with error checking

Figure 19 is a screenshot of the application; it provides an option for the tester to specify the number of events that they would like to generate, this can be any number. There are four main buttons that will be used for most of the testing, one for generating the text files, one for modifying the contents of the text files, one for renaming the text files and one for deleting the text files.

The tester application also has a display window that outputs messages to the tester, this is used for informing the user when it has started and completed any tasks, it also informs the user when the 'file counter' has been changed and what value it is at.

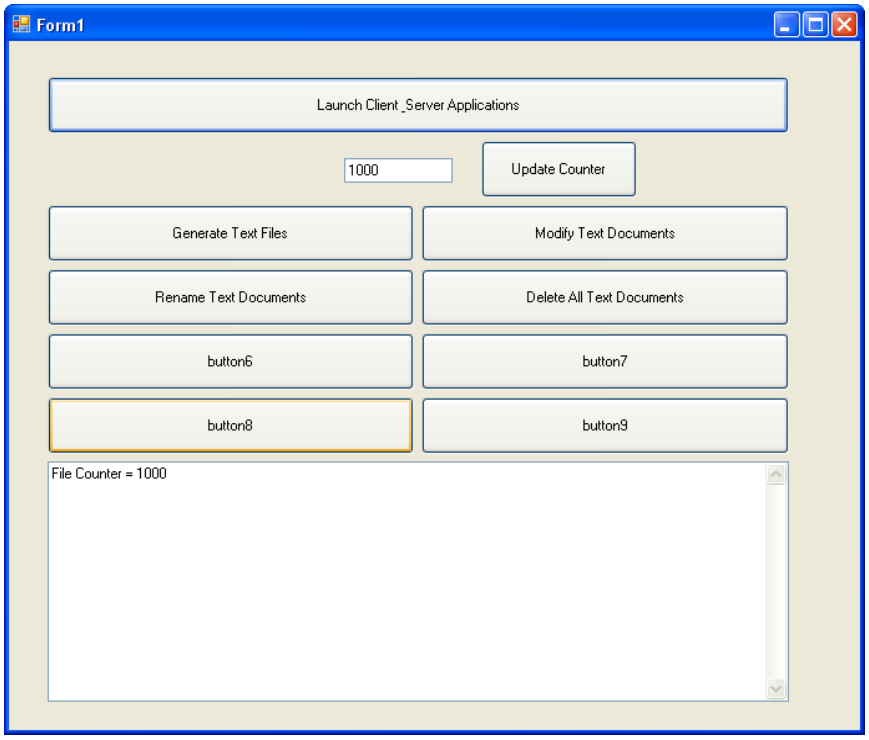


Figure 19: Tester application

4.9 Conclusion

This chapter has identified the main components of software that will be developed as part of this project. All the components of the system will be created using C# and the .NET framework, as they provide Event logging, Performance monitoring and encryption classes, which this application makes extensive use of.

Based upon the research that was done in the Literature Review and Evaluation of the Windows Event Log chapters it was decided that the application would use client-server architecture. The client would be the event logger and the server would be the data archiving system. All the communications that take place will be encrypted and all the event s that are generated will be hashed.

Also, a method for testing the performance, security and accuracy of the system was proposed. A program that would assist in the testing of the application was also designed; its main purpose is to generate a larger number of events in a very short period of time. This is to simulate the level of traffic that would be present on a corporate sever.

5 Implementation

5.1 Introduction

The three components, the event logger, the data archiving system and event viewer, that have been developed were created using C# and the .NET framework that is provided by Microsoft. These were chosen for their rapid deployment ability, and they fully support such things as XML, encryption and socket programming.

5.2 Event Logger

The event logging application will be resident on the client machine, its purpose is to monitor the file system for files within specified directories being modified. After it has captured these events it then constructs an XML sting that conatins all the ddetials of the event, this than has a hash signature tagged onto it. The whole string gets encrypted and sent to the data archiving system.

5.2.1 Connecting to Server

Figure 20 shows the code that is used for the Client to be able to establish a TCP connection to the server. The code allows for IP Addresses, Domain Names and local machine names to be entered for the server, it resolves them by using DNS.

```
Console.Write("Connecting to server... ");
Byte[] data = new byte[10240];
IPHostEntry host = Dns.GetHostEntry("localhost");
IPAddress ipAddr = host.AddressList[0];

// Specify the port to connect too.
IPEndPoint ipep = new IPEndPoint(ipAddr, 13000);

// Specify the path to watch.
Socket server = new
Socket(AddressFamily.InterNetwork,SocketType.Stream,
ProtocolType.Tcp);
try
{
  server.Connect(ipep);
}
catch (SocketException e)
{
  Console.WriteLine("Failed!");
  Console.WriteLine(e.ToString());
  Console.ReadLine();
  return;
}
```

Figure 20: Connecting to a Server

5.2.2 FileSystemWatcher

The 'FileSystemWatcher' class is part of the 'System.IO' namespace provided by .NET versions 1.1 though to the current 3.5. According to MSDN it listens to the file system change notifications and raises events when a directory, or file in a directory, changes. In Figure 21 The FileSystemWatcher is set to watch all files that are contained in the 'C:\Test\' directory. Multiple instances of FileSystemWatcher can also be run to monitor other directories, that way multiple directories can monitored using a single application.

```
// Create a new FileSystemWatcher and set its properties.
FileSystemWatcher watcher = new FileSystemWatcher();

// Specify the path to watch.
watcher.Path = "c:\\Test\\";
watcher.NotifyFilter = NotifyFilters.LastAccess |
NotifyFilters.LastWrite
  | NotifyFilters.FileName | NotifyFilters.DirectoryName;

// Specify the file and/or extension to watch.
watcher.Filter = "*.*";
watcher.IncludeSubdirectories = true;

// Add event handlers.
watcher.Changed += new FileSystemEventHandler(OnChanged);
watcher.Created += new FileSystemEventHandler(OnChanged);
watcher.Deleted += new FileSystemEventHandler(OnChanged);
watcher.Renamed += new RenamedEventHandler(OnRenamed);

// Specify the buffer size, default is 8192 (8K).
watcher.InternalBufferSize = 102400;

// Begin Watching.
watcher.EnableRaisingEvents = true;
```

Figure 21: The FileSystemWatcher class

5.2.3 Capturing Events

When a 'Renamed' event occurs it is passed to the above 'OnRenamed' class, this then wraps the various elements of the event in XML tags, this creates an 'Event Message' and includes such things as the clients time and date, the user and details about the event and the file that was renamed. This is a similar process when a file is changed. After this stage the Event Message is passed though to the HMAC section, where a hash signature value will be generated for it. Figure 22 shows this.

```
public static void OnRenamed(object source, RenamedEventArgs e)
{
  // Specify what is done when a file is renamed.
  string message = "<clientTime>" + DateTime.Now +
"</clientTime><user>" + GlobalClass.user + "</user><changeType>"
+ e.ChangeType + "</changeType><fullPath>" + e.FullPath +
"</fullPath>";

  // Pass the 'message' to the next stage.
  sendmsg(message);
}
```

Figure 22: Capturing Events

5.2.4 HMAC

Figure 23 shows the code that is used to generate the HMAC checksum. The 'key' is taken from a user input when the application is first launched; it is then passed into the following code along with the events that have been generated by the 'FileSystemWatcher' class. The resulting checksum value is then wrapped in XML tags and attached to the end of the captured events string.

```
// Declare a new instance of Ascii Ecnoding called 'encoding'.
System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();

// Convert the HMAC key to a byte value.
byte[] keyByte = encoding.GetBytes(key);

// Create a new instance of HMAC with SHA1.
HMACSHA1 hmac = new HMACSHA1(keyByte);

// Convert the message to a byte value.
byte[] messageBytes = encoding.GetBytes(chkMessage);

// Compute the hash signature.
byte[] hashmessage = hmac.ComputeHash(messageBytes);

// Update the value of the 'chkMessage' string.
chkMessage = chkMessage + "<hmacCheck>" + ByteToString(hashmessage) +
"</hmacCheck>";
```

Figure 23: Generating an HMAC hash signature

5.2.5 Encrypting

Figure 24 shows the code that is used to encrypt the Event Message, the code for it is from a program by Mathew Schlabaugh (CodeProject, 2008). The cipher text is generated by breaking down the plain text into blocks that are smaller than the key that is used to encode them. The smaller blocks are then encoded with the Servers public key, this then makes them ready to be sent to the server.

```
// The following code is from 'Public Key RSA Encryption
// in C# .NET' by Mathew John Schlabaugh.
// Available from the http://www.codeproject.com

// Specify the size of the key to be used in bits.
int dwKeySize = 1024;

// Decalre a new instance of the RSACryptoServiceProvider.
RSACryptoServiceProvider RSAProvider = new
RSACryptoServiceProvider(dwKeySize);

// Read the Servers public key from XML.
RSAProvider.FromXmlString(justPublicKey);

// Convert keysize from bits to bytes.
int keySize = dwKeySize / 8;

// Convert message to bytes.
byte[] bytes = Encoding.UTF32.GetBytes(chkMessage);

// Define a size that will be used for calculating a block size.
int maxLength = keySize - 42;
int dataLength = bytes.Length;
int iterations = dataLength / maxLength;
StringBuilder stringBuilder = new StringBuilder();
for (int i = 0; i <= iterations; i++)
{
  // Break the message apart.
  byte[] tempBytes = new byte[(dataLength - maxLength * i >
maxLength) ? maxLength : dataLength - maxLength * i];

  // Copy the smaller message into memory.
  Buffer.BlockCopy(bytes, maxLength * i, tempBytes, 0,
tempBytes.Length);

  // Encrypt the smaller message/section.
  byte[] encryptedBytes = RSAProvider.Encrypt(tempBytes, true);

  // Add the encrypted secion to the StringBuilder.
  stringBuilder.Append(Convert.ToBase64String(encryptedBytes));
}
```

Figure 24: Encrypting Data

5.2.6 Sending to Server

For the client to be able to send any data to the data archiving system it will have to converted into packet and then placed onto the TCP stream. Figure 25 shows this.

```
// Send the encrypted message to the Server.
server.Send(Encoding.ASCII.GetBytes(encryptedMessage));

// Specify the packet size.
Byte[] data = new byte[10240];

// Receive the acknowledgement from the Server.
int recv = server.Receive(data);

// Convert the recieved message to a string.
String stringData = Encoding.ASCII.GetString(data, 0, recv);
```

Figure 25: Sending Data to a Server

5.3 Data Archiving System

5.3.1 Listening for a Connection

The data archiving system is in an always listening state until a connection has been made, Figure 26 shows that this one is listening on port 13000 for an incoming TCP connection. When a Client connects it takes records some of its details, and it then generates an RSA Key Pair, which is then sent t the client.

```
// Lisetn for a connection on port 13000.
IPEndPoint ipep = new IPEndPoint(IPAddress.Any,13000);

// Create a new socket for the connection.
Socket newsock = new
Socket(AddressFamily.InterNetwork,SocketType.Stream,
ProtocolType.Tcp);

// Bind the socket to the connection.
newsock.Bind(ipep);

// Place the socket into a listening state.
newsock.Listen(10);
Console.WriteLine("Waiting for a client...");
Socket client = newsock.Accept();
IPEndPoint clientep =(IPEndPoint)client.RemoteEndPoint;

// Display a message when a client has connected.
Console.WriteLine("Connected with {0} at port {1}",clientep.Address,
clientep.Port);
```

Figure 26: Server Listening for a Connection

5.3.2 Generating Keys

Figure 27 shows the routine that was used to generate both the public and private RSA keys, these keys are then exported to an XML string, the public key gets sent to the Client and the Private key stays with the data archiving system.

```
// Specify the size of the key that will be generated.
int dwKeySize = 1024;

// Create a new instance of RSACryptoServiceProvider to generate
// public and private key data using the specified key size.
RSACryptoServiceProvider RSAProvider = new
RSACryptoServiceProvider(dwKeySize);

// Convert the public and private keys into XML format.
string publicAndPrivateKeys = RSAProvider.ToXmlString(true);

// Convert the public key into XML format.
string justPublicKey = RSAProvider.ToXmlString(false);
```

Figure 27: Generating an RSA Key Pair

5.3.3 Receiving Message

The data archiving system will continue to listen and accept messages (see Figure 28) that have been received from the client until the client disconnects. All the messages that are received are passed through to the decrypt class, where the data archiving systems private key is applied to the cipher text and thus decrypting it back to plain text.

```
while (true)
{
  // Specify the size of the packet.
  data = new byte[10240];

  // Accept the packet from the Client.
  recv = client.Receive(data);

  // If the Client disconnects then exit.
  if (recv == 0)
    break;

  // Send the packet back to the Client.
  client.Send(data, recv, SocketFlags.None);
}
```

Figure 28: Receiving a Message

5.3.4 Decrypting Message

Figure 29 shows the code that is used to decrypt the received message; the code for it is from a program by Mathew Schlabaugh on the codeproject.com which can be found at (http://www.codeproject.com/KB/security/RSACryptoPad.aspx).

```
// Convert the encrypted message back into a string
string encryptedMessage = Encoding.ASCII.GetString(data, 0, recv);

// Use the private key to decrypt the message
RSAProvider.FromXmlString(publicAndPrivateKeys);
int base64BlockSize = ((dwKeySize / 8) % 3 != 0) ? (((dwKeySize / 8)
/ 3) * 4) + 4 : ((dwKeySize / 8) / 3) * 4;

// Break the message apart into blocks.
int iterations = encryptedMessage.Length / base64BlockSize;
ArrayList arrayList = new ArrayList();
for (int i = 0; i < iterations ; i++)
{
  byte[] encryptedBytes =
Convert.FromBase64String(encryptedMessage.Substring(base64BlockSize *
i, base64BlockSize));
  arrayList.AddRange(RSAProvider.Decrypt(encryptedBytes, true));
}

// Convert the decrypted message to a string.
string decryptedMessage =
Encoding.UTF32.GetString(arrayList.ToArray(Type.GetType("System.Byte"
)) as byte[]);
decryptedMessage = "<ServerTime>" + DateTime.Now + "</ServerTime>" +
decryptedMessage;
```

Figure 29: Decrypting a message

5.3.5 Saving To Disk

Figure 30 shows the code that is used to write the Event Message to disk, firstly it tries to open the file to add text to it, and if it is unsuccessful it creates a new file and then adds the plain text XML string to it.

```
// Create an new instance of a textwriter called 'tsw'.
TextWriter tsw;

// Attempt to open the log file to add text, if it fails then create
// a new one.
try
{
  tsw = File.AppendText("C:\\EvtLog2.log");
}
catch
{
  tsw = new StreamWriter(@"C:\\EvtLog2.log");
}

// Write the decrypted message to file.
tsw.WriteLine(decryptedMessage);

// Close the file.
tsw.Close();
```

Figure 30: Saving Event Data to Disk

5.4 Event Log Reader

This application will be used to be able to read the contents of the log file that has been created, it will allow the user to be able to read through the entries in the Event Log file one at a time, similar to a flat database. The Log Reader will also perform the authentication of the Event Log data, for this to be achieved the user will be required to enter the exact same key that was used to generate the original HMAC value. Based upon the key that was entered, an HMAC value will be generated on the same information as the original checksum, as long as the data and key are the same as the original values, when these are compared they should produce the same HMAC value.

5.4.1 Opening File

Figure 31 shows the code that was used to read the Event Log file from the disk, it does this line by line. At the same time it copies the information from the line that it has just read to an array, it then moves onto the next line in the file and repeats the process. One point to note is that there is also a check in here to ensure that only the lines in the log file that start with '<ServerTime>' are copied to the array. This is done

because the server writes a line to the file when a connection is established and another line when the connection is terminated; at this point in time this project is only interested in the data that is being sent from the client.

```
// Declare new instance of StreamReader and tell it the filename.
StreamReader sr = new StreamReader("c:\\EvtLog2.log");

// Decalare local variable.
string line;

// Repeat the following until all lines are read from file.
while ((line = sr.ReadLine()) != null)
{
  // Only copy the lines that contain <ServerTime>.
  if(line.Contains("<ServerTime>"))
  {
    // Add the <ServerTime> lines to the array called 'lines'.
    lines.Add(line);
  }
}
// Close the file.
sr.Close();
```

Figure 31: Reading a file

5.4.2 Decoding XML

To convert the flat XML strings that are contained within the Log file a simple piece of code was written that could extract the data. Figure 32 shows this for extracting the data from the <serverTime> tags. First it calculates the starting point of the required data by working out the size of the opening tag. Then it calculates the end point of the data based upon the length of the closing tag. It then displays the data on the screen.

```
// Copy the currently selected array item to string1.
string string1 = Convert.ToString(lines[arrayItem]);

// Define the start tag.
string myString1 = "<ServerTime>";

// Define the end tag.
string myString2 = "</ServerTime>";

// Calculate the starting and ending points of the data contined
// within the start and end tags.
serverTimeLbl.Text = string1.Substring((string1.IndexOf(myString1) +
myString1.Length), (string1.IndexOf(myString2) -
(string1.IndexOf(myString1) + myString1.Length)));

// Repeat this process for all the tags.
```

Figure 32: Dividing up an XML string

5.4.3 HMAC Checksum

Figure 33 shows the code that was used to calculate the second HMAC value. This key value is calculated on the key that the user has entered in the viewer. It is then displayed on the screen. Both the 'new' and 'old' values are then compared to see if they match. Depending upon the results of the comparison an appropriate message is displayed to the user. This method works because with HMAC so long as the key and data remain constant the output will be the same.

```
// Declare a anew instance of Ascii Encoding.
System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();
byte[] keyByte = encoding.GetBytes(hmacTxt.Text);

// Declare new instance of HMAC.
HMACSHA1 hmac = new HMACSHA1(keyByte);

// Convert the message to be checked into bytes.
byte[] messageBytes = encoding.GetBytes(chkMessage);

// Calculate the HMAC value of the message.
byte[] hashmessage = hmac.ComputeHash(messageBytes);

// Display the 'new' hmac value on screen.
hmacCheckLbl.Text = ByteToString(hashmessage);

// Compare the values of the one on file to the new one.
if (hmacCheckLbl.Text == hmacLbl.Text)
{
  // If they are the same.
  hmacCheckLbl.Text += " : Hmac Passed!";
}
Else
{
  // If the are different.
  hmacCheckLbl.Text += " : Hmac Failed!";
}
```

Figure 33: Checking the hash signature

5.5 Conclusions

The data that is being transferred to the Data Archiving System is successfully being encrypted using RSA. This is regarded as a secure form of encryption that uses complex mathematics to encrypt and decrypt data. There are also two keys involved, one public key and one private key. The public key can only be used for encrypting a message and is given to the person that is sending the message. Only the paired private key is able to decrypt that message.

All three components of the application make full use of HMAC signature hashing as method for authenticating the messages that have been received from the event logger

by the data archiving system, and also by the event viewer when reading the contents of the event log.

Unfortunately due to time constraints, these components of the application were not as fully developed was they could have been. The original design of the Data Archiving System specification was to allow for both of the Event Logging Application and Event Viewer Application to be able to remotely access the event log that was stored on the Server, unfortunately there is no support within the Server Application to allow for a remote connection to the Server, it will however run on the Server without any problems.

It had been originally planed that the encryption between the Client and Server could also be done using synchronous encryption with the secret key being exchanged using asynchronous encryption. The communications that both the Client and Server use is only asynchronous encryption; this will obviously cause problems with the performance.

Also, the Server does not run in a finite state, that is after a Client has terminated its connection to the Server, the Server application will shutdown. It could have been written so that the Server returns back to its original listening state.

Another unachieved goal of the Event Viewer Application was for it to be able to display a list of all the events that have happened and for it to allow the user to be able to browse for a specific event.

Apart from these minor short comings the overall concept of having an Event Logger that securely stores its Event Logs off site using a remote connection has been achieved. If time permits a maintenance phase could be used to try and address some of these issues.

6 Evaluation

6.1 Introduction

Initially the application is tested to identify the correctness and completeness of the developed software, any faults or problems that are identified will be dealt with at the maintenance stage. After modifications have been made and documented, the application will then undergo further testing that will measure the burden that it places on the system. The application will be stress tested; a large number of events will be generated to check if the application is able to detect all of them. An attempt will also be made to see if it is possible to try and calculate the HMAC key using a brute force algorithm. Based upon the results of this test it will be possible to recommend a suitable sized key.

6.2 Initial Testing

This phase was to test the initial design that had been implemented. Its purpose was to identify if the Event Logging Application could successfully capture a large amount of information without missing any events and to identify any points in the system that needed to be modified beyond their original design specification.

The testing was carried out using the automatic testing application that had been developed. Initially 1,000 events were generated and captured, it had been noticed that there was a delay in getting all these events written to file by the Server. The same test was then carried out several more times with the number of events increasing by 1,000 each time. One point to note is that the application became unresponsive when 5,000 events were generated.

After the Initial Testing phase was completed a serious flaw in the application was detected, namely a large number of events being generated caused a buffer overflow in the .Net FileSystemWatcher class. The MSDN website says that “If there are many changes in a short time, the buffer can overflow. This causes the component to lose track of changes in the directory, and it will only provide blanket notification.”

(Microsoft, 2008). This caused the FileSystemWatcher to crash and it then failed to capture any other events, until the application was restarted.

It had been suspected that the buffer overflow problem may have been caused by either the size of the buffer, the HMAC tagging or the asymmetric encryption that was happening after the events were captured; Figure 34 explains the process.

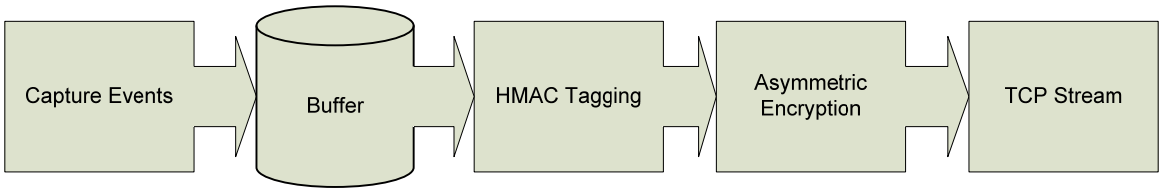


Figure 34: Data flow

Performance tests were then carried out on each of the individual components; this initially consisted of increasing the size of the buffer, which made very little difference when dealing with upwards of 5,000 events. A test was created that would have the FileSystemWatcher capture events and send them directly to the TCP Stream, thus bypassing the HMAC and Asymmetric Encryption. It had then been noted that the previous buffer overflow problem had gone, even when the buffer size was set back to its default value of 8K it still managed to successfully capture 20,000 events. After this successful test the HMAC tagging was added back in, this too did not hinder the performance of the application, which left only the asymmetric encryption process to be responsible.

6.3 Maintenance

Based upon the preliminary set of test results, the Event Logging Client and Server applications were heavily modified to improve the encryption process. This involved the use of symmetric encryption for handling the transfer of the events from the Client to the Server. The shared key that the symmetric encryption uses is transferred from the Client to the Server using asymmetric encryption.

This brings in several new stages for the Client Server Communication Protocol that was originally described in the Design chapter. Figure 35 explains the new process:

Figure 35: Updated Client-Server communications

6.4 Experiment 1: Performance

This experiment will be used to measure the performance of the application; it will also measure the load that it places on the system.

A simple application (see Figure 36) was developed that would poll the processor every 100 milliseconds and return its value as a percentage. This data along with the current system time is then displayed on screen; the system time is used so that the data can be cross referenced to the event log data. An option is given for the user to save this information to a text file, so that it can easily be imported into a spreadsheet application to produce a graphical analysis of the results.

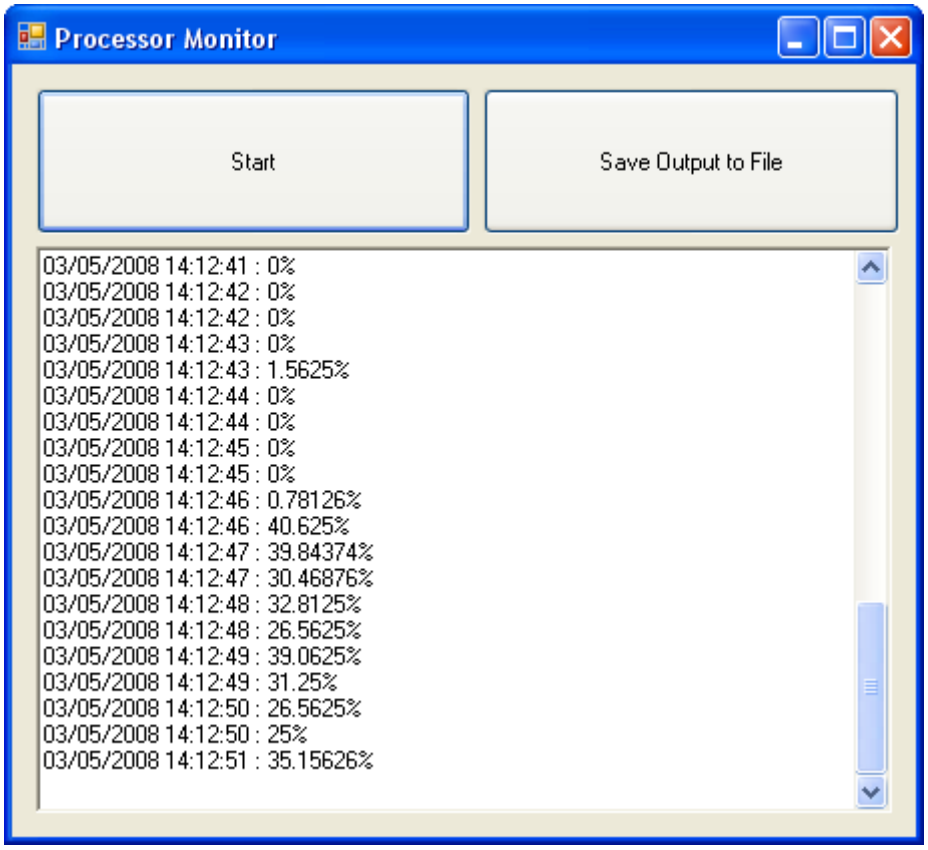


Figure 36: Processor monitor application

First is to create a baseline without any encryption happening, this test will be run to measure the load being placed on the processor when 1,000, 5,000, 10,000 and 20,000 files are being created. Each test will run over a 30 second interval, this will provide ample time to complete the test and will also visually depict the information more clearly. Figures 37, 38, 39 and 40 show these results. As these experiments have been carried out on a quad core processor the pink line represents a single core.

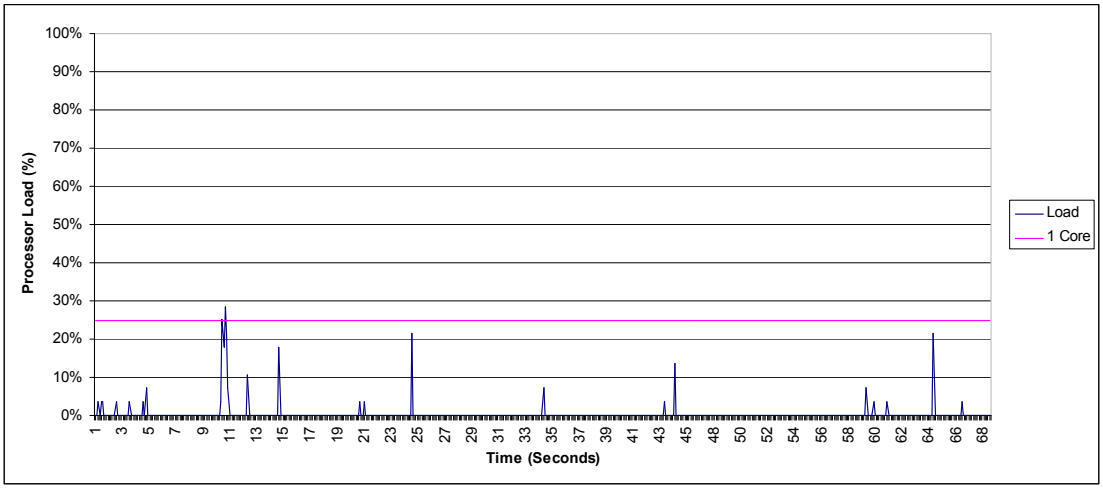


Figure 37: Generating 1,000 files with no encryption

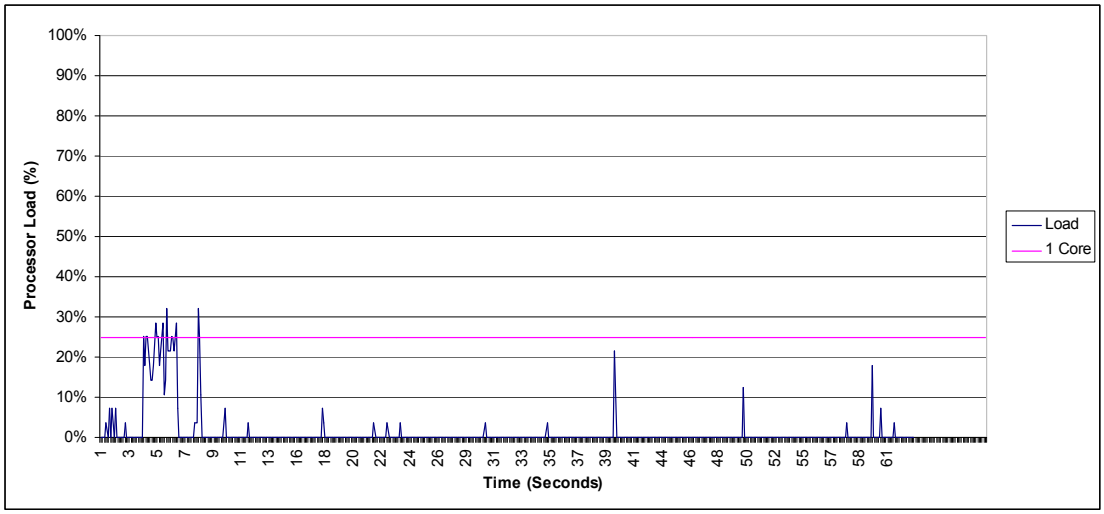


Figure 38: Generating 5,000 files with no encryption

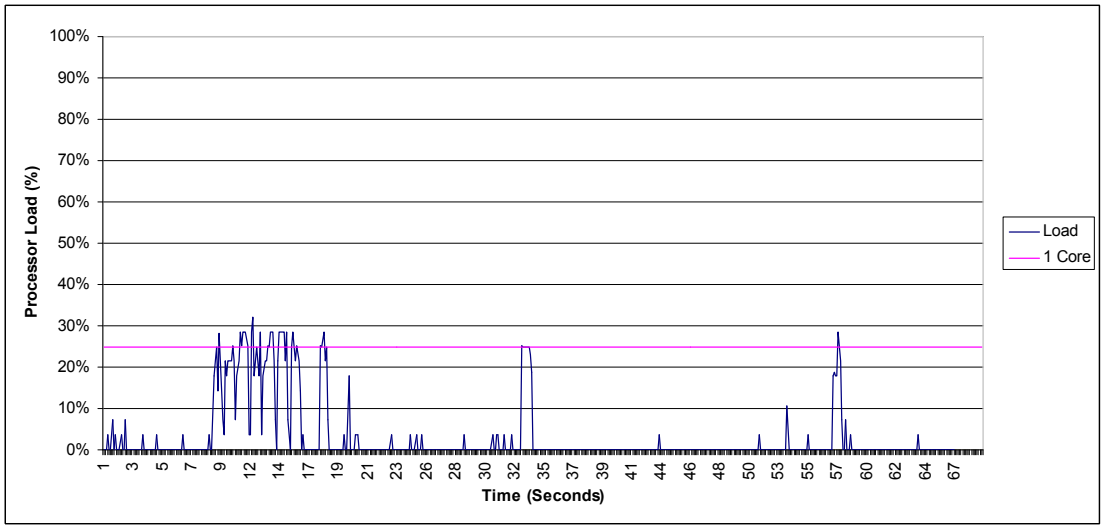


Figure 39: Generating 10,000 files with no encryption

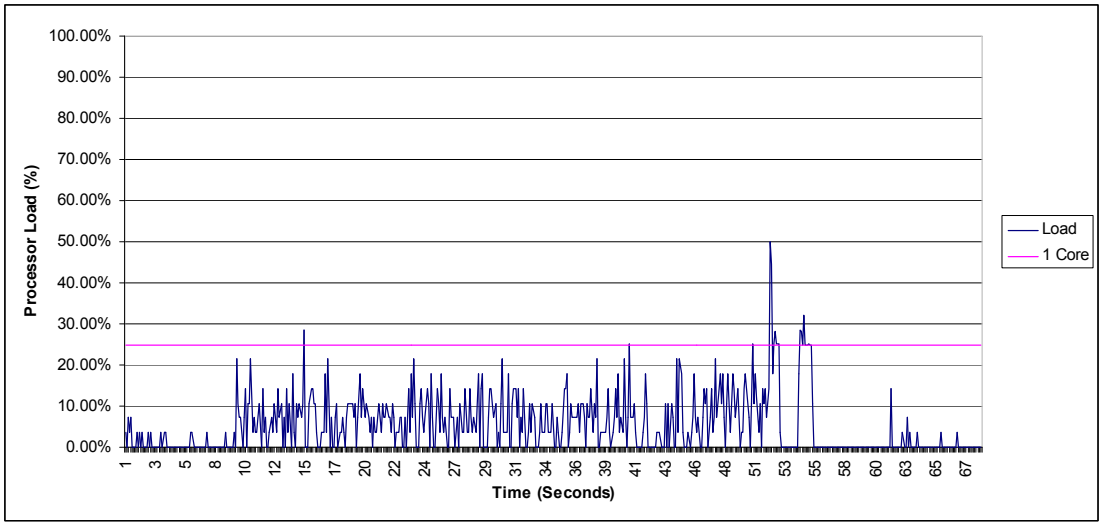


Figure 40: Generating 20,000 files with no encryption

Next the exact same tests are rerun, but this time the Client and Server are used so the load that the symmetric encryption places on the system can be measured, once again the following graphs are of 1,000, 5,000, 10,000 and 20,000 being created. Figures 41, 42, 43 and 44 show these results.

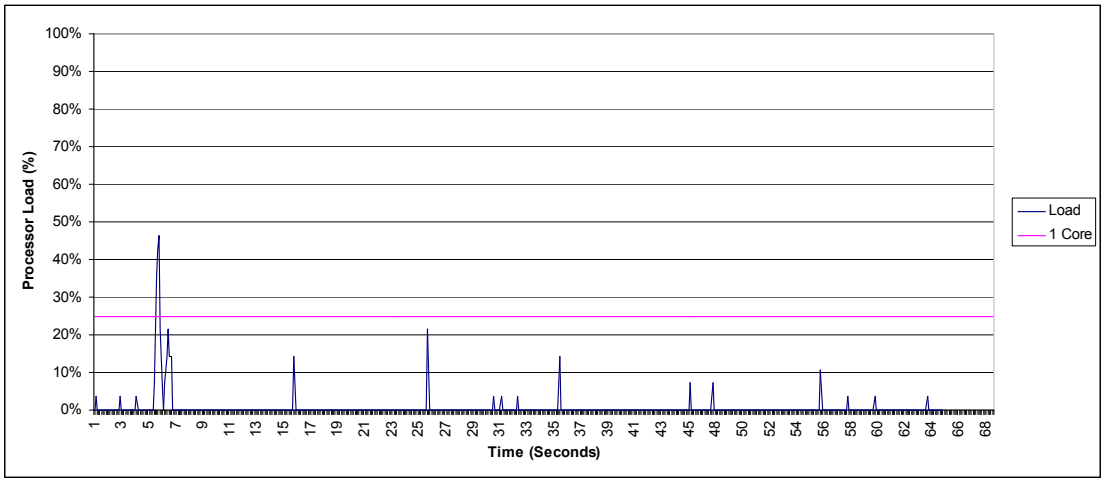


Figure 41: Generating 1,000 files with symmetric encryption

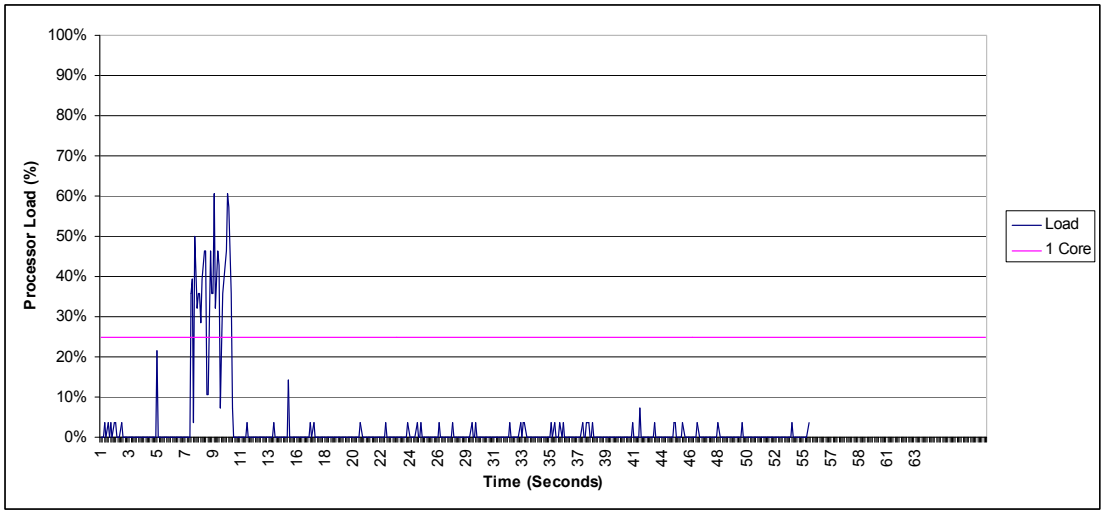


Figure 42: Generating 5,000 files with symmetric encryption

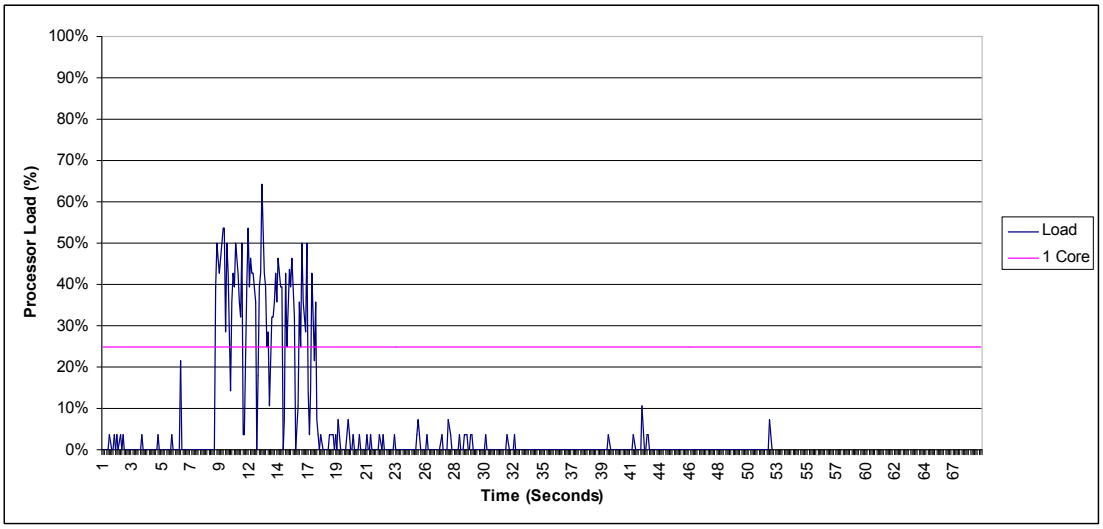


Figure 43: Generating 10,000 files with symmetric encryption

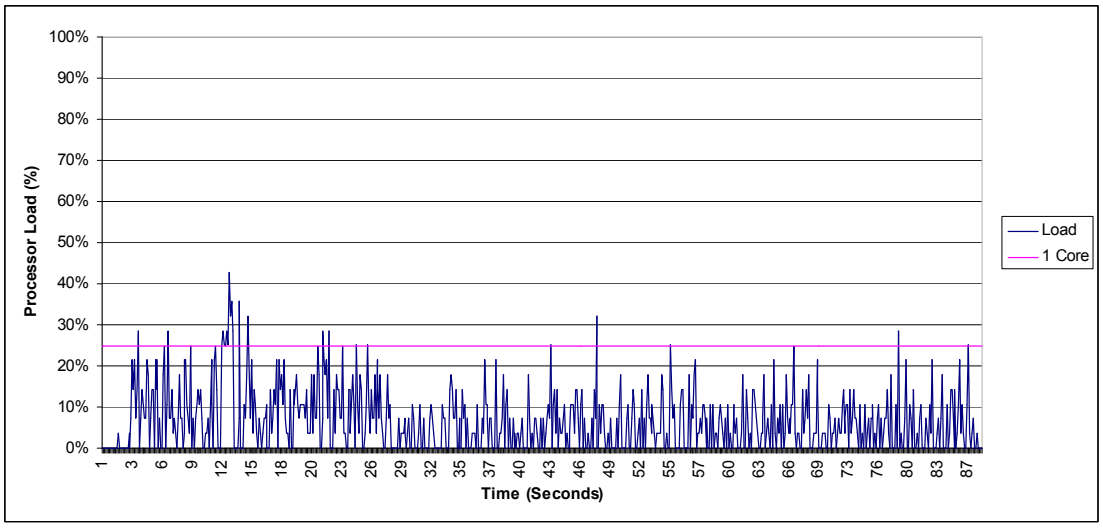


Figure 44: Generating 20,000 files with symmetric encryption

From these results it is also possible to calculate the amount of time it takes to produce these files using no encryption, symmetric encryption and asymmetric encryption. Figure 45 shows the comparison in the length of time it takes to encrypt events, with symmetric encryption, that are generated when files are created on a system.

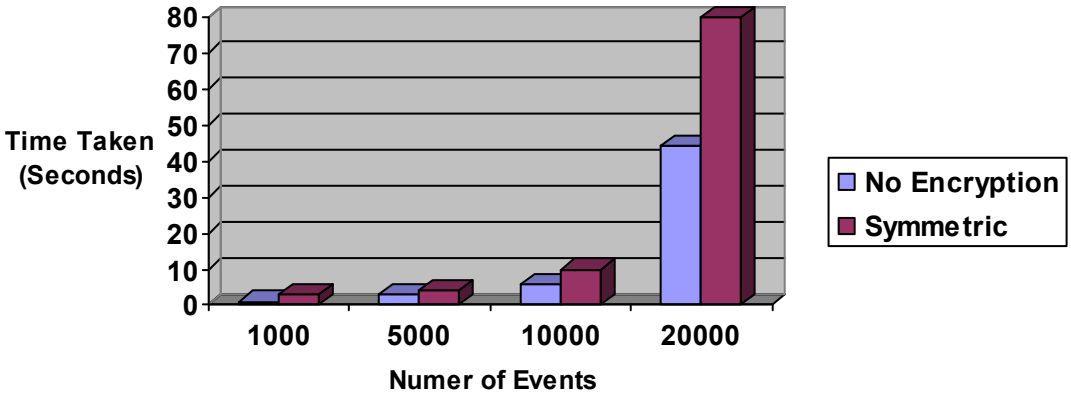


Figure 45: Symmetric encryption comparison

The above graph shows that it approximately takes 50% longer when using symmetric encryption as opposed to not encrypting anything. These results can then be compared to the length of time it takes using asymmetric encryption. Figure 46 compares the time taken to create the same files, but this time the events are encrypted with asymmetric encryption.

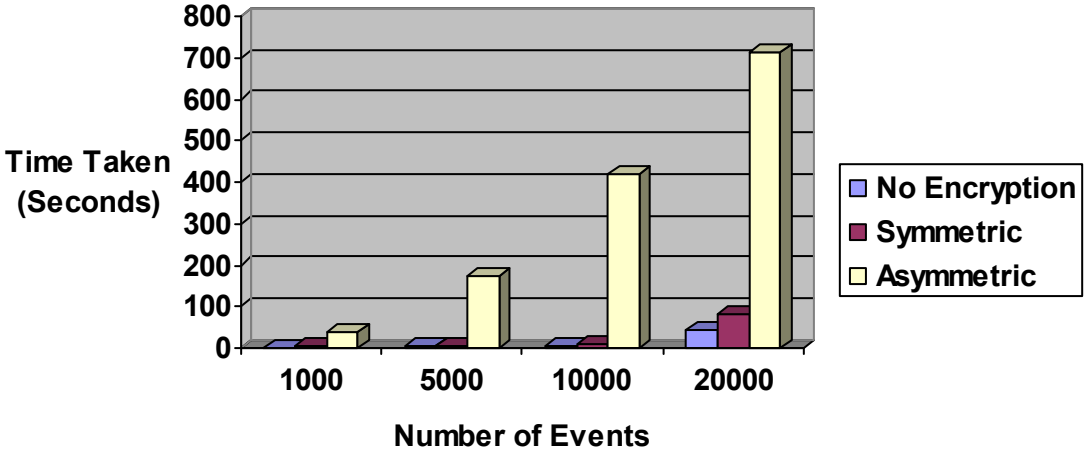


Figure 46: Asymmetric encryption comparison

The above graph shows that using asymmetric encryption is approximately 8 times slower than symmetric encryption and 16 times slower than having no encryption at all. This justifies changing the project to use symmetric encryption for transferring the events to the data archiving system.

6.5 Experiment 2: Accuracy

This experiment will be used to check the accuracy of the application, to ensure that all the events that are being generated are actually being captured by the application. The tester application was slightly modified to allow for an event log file to be modified, it now randomly modifies the 'SecEvent.txt' file. For the purpose of this experiment a text file that is contained within a different directory to the data files, that are being created, is used to represent the event log file. The directory structure is represented by Figure 47.

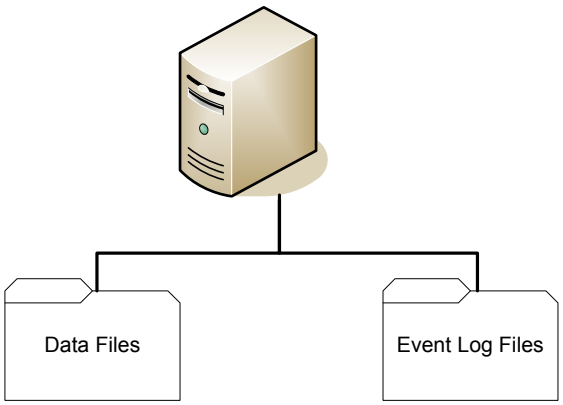


Figure 47: Test directory structure

The tester application was set up to create 1,000 files and, at some random point during this time, to modify the 'SecEvent.txt' file. Figure 48 shows in its display box that the 'File Counter' is set to 1000 and that the 'SecEvent.txt' would be modified after the 426th file was created.

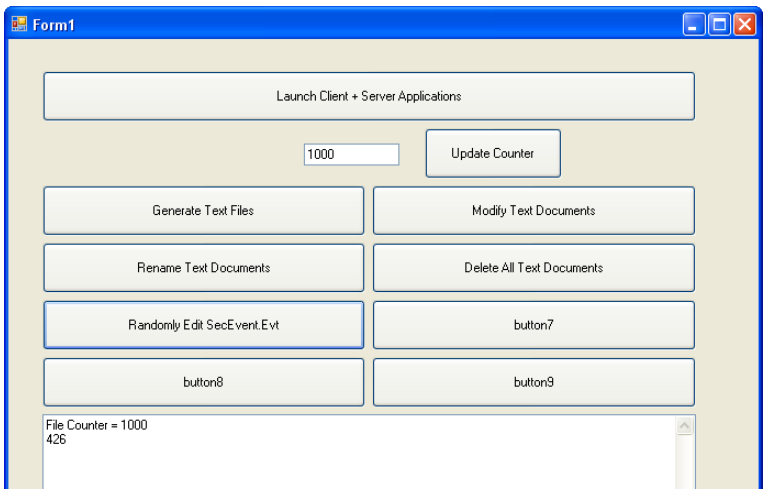


Figure 48: Tester application generating random event

Using the Event Log Viewer application (See Figure 49), it showed that the event log file had 1001 entries in it, and after browsing through the event viewer it displayed that the 'SecEvent.txt' file had been modified. This also clearly displays the HMAC check working, note that the original key is entered at the top of the window.

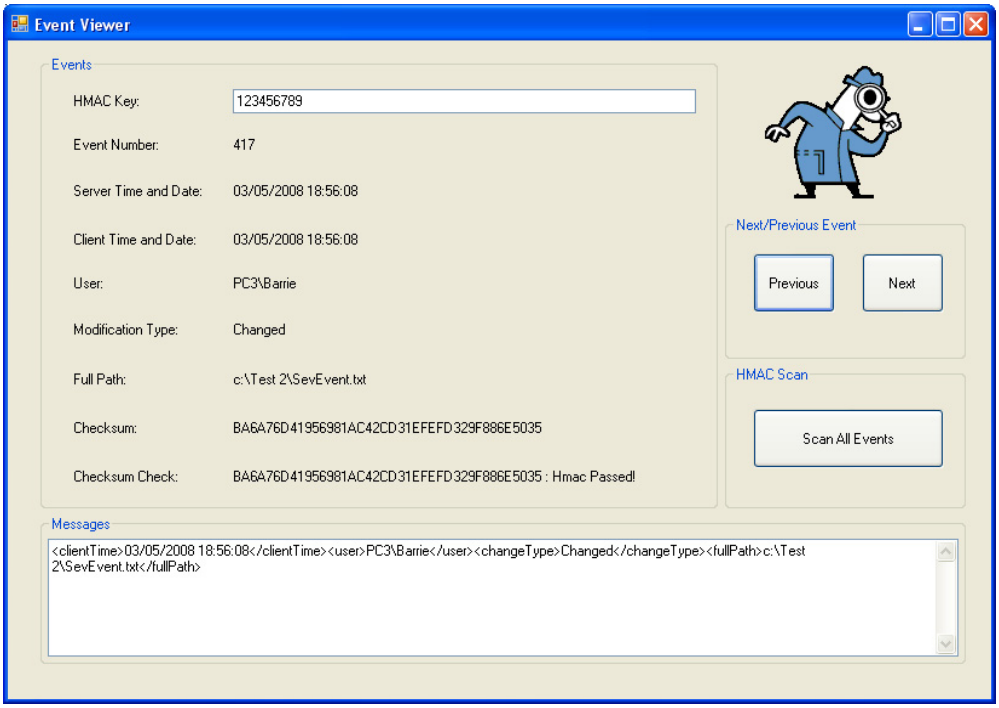


Figure 49: Event viewer application

This experiment was then rerun with the previous benchmark figures of 1,000, 5,000, 10,000 and 20,000 files being created. All of the tests provided positive results. Figure 50 is an extract from the event log file; it shows that it has successfully captured the event of the 'SecEvent.txt' file being modified.

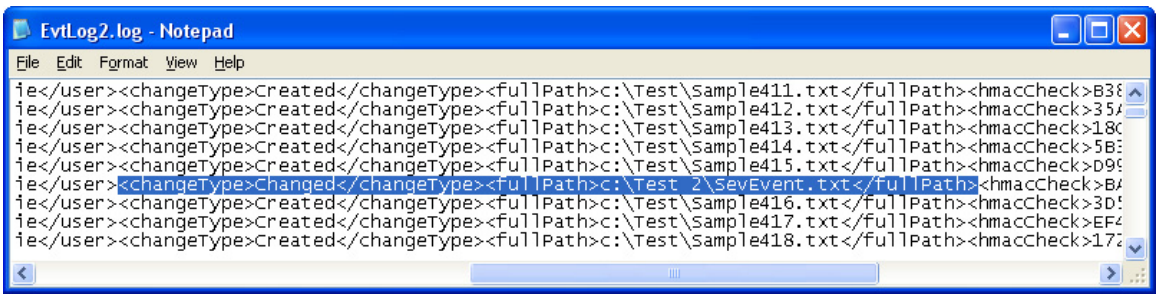


Figure 50: Custom event log

6.6 Experiment 3: Security

This experiment will be used to try and measure the whether or not the application meets the security requirements. This will be done by checking the HMAC checksum that is tagged onto each event and also the synchronous and asynchronous encryption. When an event is generated it produces an XML string which (see Figure 51) contains the following information:

Client Time and Date	Computer and Username	The Type of Event	The Full Path of the File

Figure 51: Client XML string without HMAC

This information is then passed into the HMAC checker, which calculates a value that is based upon this information and a predetermined key. It will produce a value that is tagged onto the end of the XML string. The XML string will now look like Figure 52.

Client Time and Date	Computer and Username	The Type of Event	The Full Path of the File	HMAC Check

Figure 52: Client XML string with HMAC

To check that the HAMC value that is being produced is valid, the original XML string along with the original key will be put through a separate hash calculator called 'HashCalc' (http://www.slavasoft.com/hashcalc/index.htm) , this will produce an HMAC value which should be the same as the one that is tagged on the end of the XML string. Figure 53 shows the data that was used to generate the HMAC hash.

Original Key:	123456789
Original String:	<clientTime>30/04/2008 11:19:28</clientTime> <user>PC3\Barrie</user><changeType>Created</changeType> <fullPath>c:\Test\Sample4591.txt</fullPath>
HMAC Output:	55B7839A807951FED96CD0927511962280258793

Figure 53: HMAC test data

As it can be seen from Figure 54, both of the HMAC values are identical, which means that the Event Logging Application is correctly generating an HMAC checksum. This was then double checked with an online Hash Calculator, the exact same details were inputted and this was the output.

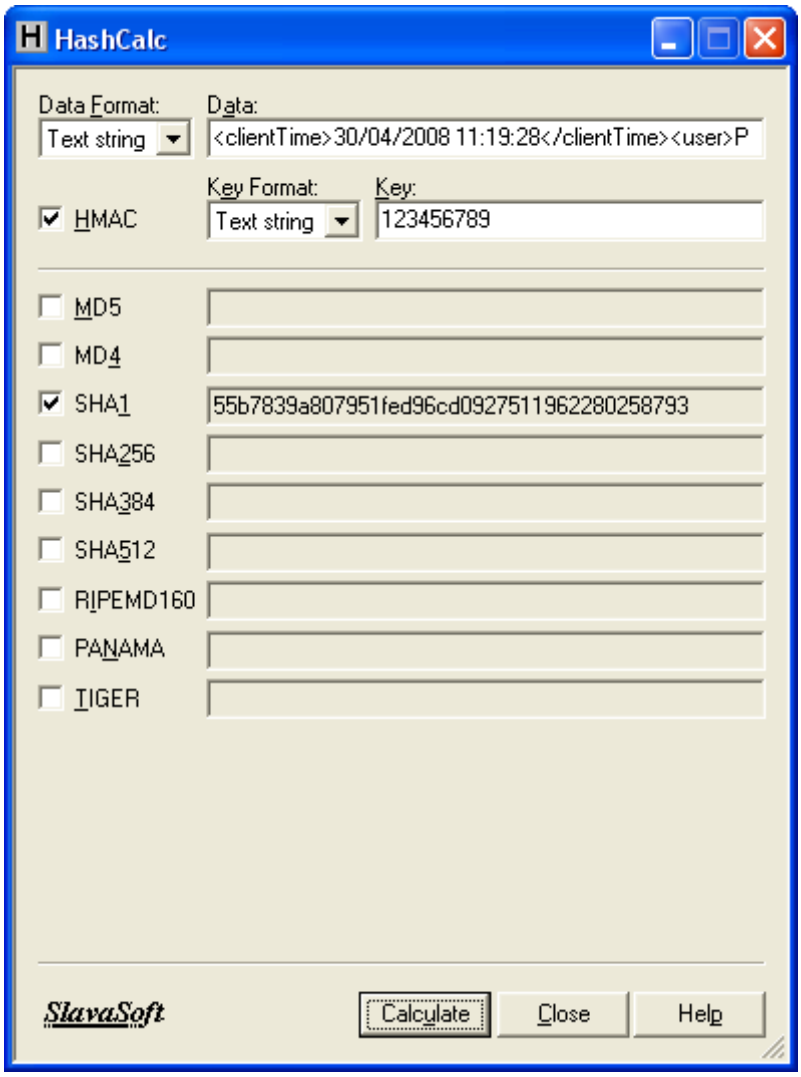


Figure 54: HashCalc Screenshot

Figure 55, which is a screenshot from iFrame.in (2008) confirms that the HMAC function is working correctly. As before the exact same details from Figure 53 were inputted and it produced the same hash signature that is displayed in Figure 53 and Figure 54.

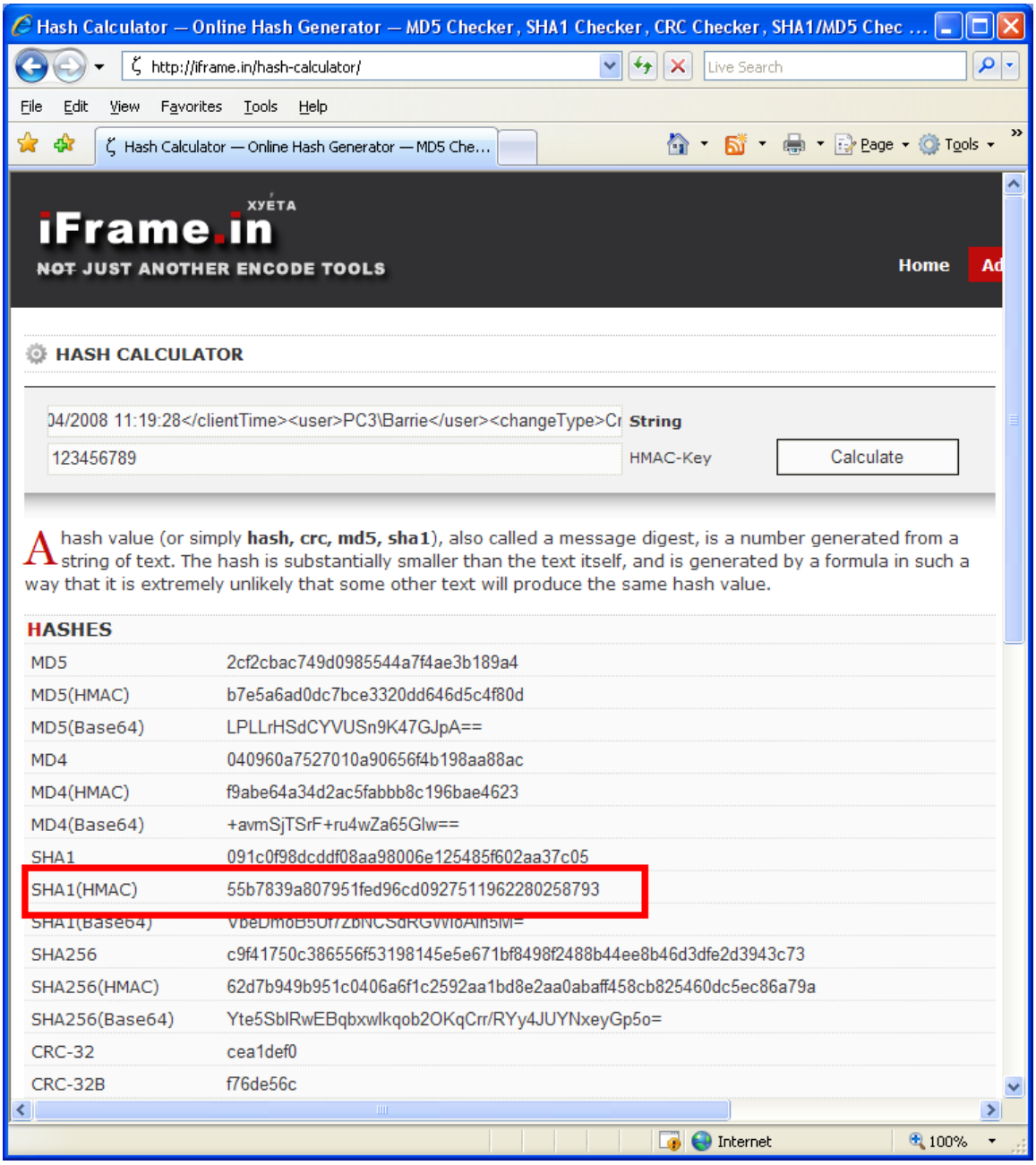


Figure 55: iFrame.in Hash Calculator

To further check the security of HMAC a brute force cracker was written that would systematically check for the key that was used to create the hash signature. The brute force application starts checking all the possible single character keys from '0' to 'z'. There are 75 different characters, so there are 75 different possible single character keys. The program then checks all the two character keys '00, 01, 02…' through to 'zz', for this there are 5,625 different combinations. Next it checks the for all the three character keys '000, 001, 002…' until 'zzz'. The full code for this can be found in Appendix H.

For testing purposes it was initially limited to a 3 character key. The information in Figure 56 was used. As it can be seen from Figure 57, after 20 seconds the brute force cracker program successfully managed to find the key that was used to generate the hash signature.

Original Key:	zzz
Original String:	test
Hash Signature:	0f0203fc9e5902739e7fb6434e7619c0caecc693

Figure 56: Three character key HMAC test data

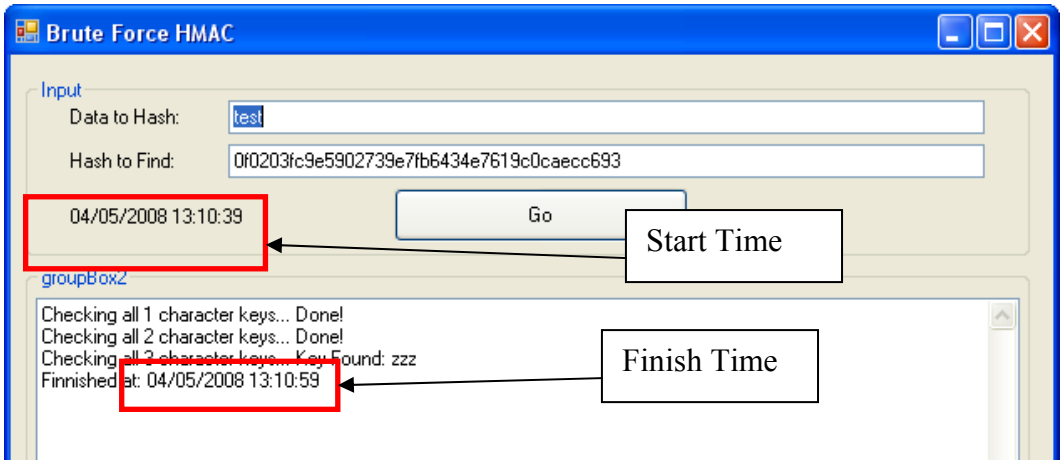


Figure 57: HMAC brute force application

This same test was again repeated with a 4 character key. Figure 58 shows the details that were used. Figure 59 shows that it took 25 minutes to decode this one.

Original Key:	zzzz
Original String:	test
Hash Signature:	f1e16dce22437e68581b664331cb2674ab2182fe

Figure 58: Found four character key

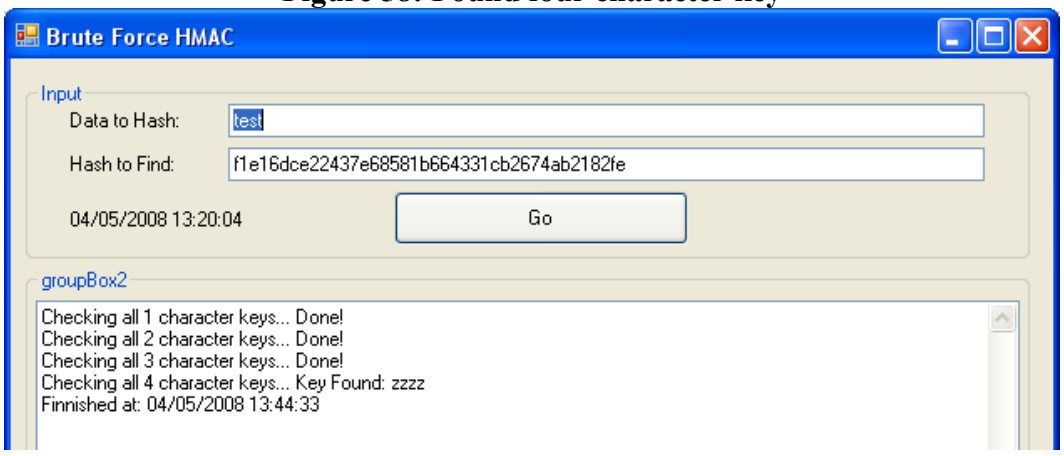


Figure 59: Found four character key

The character set being used is 75 characters long, ranging from '0' to 'z'. Using this from the previous experiments it is possible to calculate the length of time that it should take to create all the possible different hash signatures based upon the key length. Figure 60 shows that the longer the key is, the longer it would take to find it out use a brute force attack.

Key Length	Number of Keys	Time (Approx)
1	$75^1 = 75$	3.6 Milliseconds
2	$75^2 = 5{,}625$	260 Milliseconds
3	$75^3 = 421{,}875$	20 Seconds
4	$75^4 = 31{,}640{,}625$	25 Minutes
5	$75^5 = 2{,}373{,}046{,}875$	31.25 Hours
6	$75^6 = 177{,}978{,}515{,}625$	97.66 Days
7	$75^7 = 13{,}348{,}388{,}671{,}875$	20.07 Years
8	$75^8 = 1{,}001{,}129{,}150{,}390{,}625$	1,504 Years
9	$75^9 = 75{,}084{,}686{,}279{,}296{,}875$	112,873 Years
10	$75^{10} = 5{,}631{,}351{,}470{,}947{,}265{,}625$	8,465,492 Years

Figure 60: HMAC key entropy

Based on these figures it is possible to calculate that 21,093 keys are being processed by the computer every second. This is relevant because a word-list or dictionary-attack could be performed with great ease. The second edition of the Oxford English Dictionary (OED, 2008) contained 291,500 definitions, which would take just under 14 seconds to try all of them. Even if the dictionary was 100 times bigger, it would still only take about 24 minutes to break a password that was in it. This means that random keys should be used, not ones that are based on proper words or names.

6.7 Experiment 4: Conformance

This test is to ensure that the application developed meets the required specifications as set out in the Design chapter; it will include a rerun of the original tests that were performed in Chapter 3 – Windows Event Log. The application will be deemed to be correctly working if it can successfully identify who and when the windows event log has been modified.

For this test a simple application will be created and installed as a service on a Microsoft Server 2003 environment, this will be used to call the Client application. It will be run under VMWare with a second Server 2003 that will act as the log server. A Microsoft Windows XP client will be used to generate the traffic of files being modified via the use of a mapped drive to the File Server.

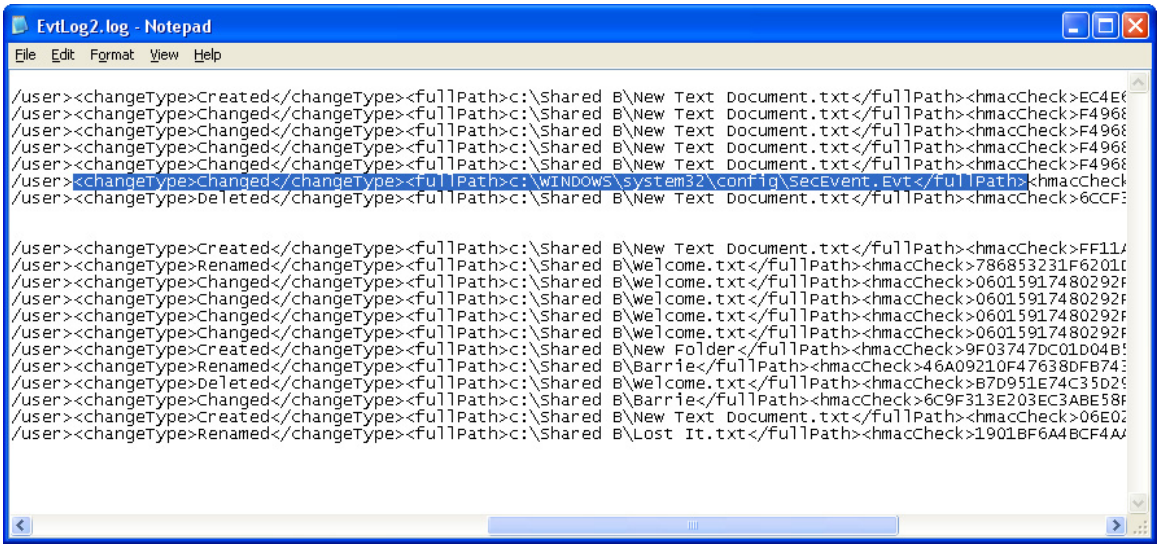


Figure 61: Custom event log showing 'SecEvent.Evt'

As it can be seen from the above screenshot, the application successfully managed to monitor the windows event log folder and the shared data folder. It was noted that the application was not properly capturing the username that caused the event, it was however, capturing the username of the user that was currently logged onto the server, in this case the administrator.

6.8 Conclusions

Based upon the testing that was carried out it was initially noted that the use of asynchronous encryption, to encode a large number of events, caused a serious drain on the system. When both the synchronous and asynchronous encryption techniques were compared to each other it was found that the synchronous encryption was 800% faster than the asynchronous one. These results lead to the modification of the application to make use of asynchronous encryption only for exchanging the synchronous shared key.

The accuracy of the application was measured by generating a large number of events on the system, when using synchronous encryption it handled them with ease. Before

it was modified the asynchronous encryption caused a buffer overflow which in turn caused the application to stop capturing events. There still remains the problem of the possibility of a man-in-the-middle attack.

The HMAC hash is prone to a brute force attack and even more prone to a dictionary-based attack. It had been found that 21,093 keys could be checked every second. The only way to minimise the risks of brute force and dictionary attacks would be to use a large random key, which should be changed at a predetermined interval, this could be either monthly or yearly. Another problem that exists is the credibility of historical events in the event log could become questionable. Perhaps these could be better protected by archiving them using encryption.

Overall the system successfully managed to capture when the 'SecEvent.txt' file was being modified while a large number of other events were happening at the same time. These events were then securely stored on the Data Archiving system.

7 Conclusions and Future Work

7.1 Conclusions

This dissertation looked at the problems that exist with the Windows event log. It had been found that digital forensic investigators use the Windows event log to create a timeline of events of what has happened on a computer system. Unfortunately the Windows event log can easily be changed to conceal or remove key pieces of information. Even within a server based environment, where the event logging servers may be kept under lock and key they are still susceptible to abuse, as someone within the company will have the key to the server.

It was pointed out that the time stamping of the events can be unreliable, so a central time stamping system needs to be used. It has also been suggested that the logs should be stored off site and by a company in a 'write once read many' environment and that they would not give physical access to the servers, and that the data on these servers contains a checksum to help validate the integrity of the logs and finally the logs should also be encrypted before they are sent to the log file storage company.

An investigation into the weaknesses of the Windows event log found that it lacked any form of security, and that it was possible to make changes to the data contained within it. Times, dates, usernames and computer names could all be easily changed. However, all of the changes involved having access to the event log files. It had been discovered that .NET has a class, the FileSystemWatcher, that watches the file system to see if any files have been modified. This was used in the application that was developed. After the application was developed, it was thoroughly tested and based upon these preliminary tests it was modified to improve its performance and functionality. More tests were then carried out, the results of which were compared the results to the preliminary tests. It had been found that using synchronous encryption was 800% faster than asynchronous encryption. And that a good strong key used for producing an HMAC hash would be hard to break.

7.2 Future Work

Overall the application does comply with the original design specification that was set out in Chapter 4; unfortunately there are a few flaws which would need to be addressed before the application could be fully released. The first flaw was stated in the conformance testing. The application does not capture the username of the person that is modifying the files, but it does however capture the username of the person that is currently logged onto the server.

The second flaw is in the current design of the Client application; currently when the application starts it prompts the user/administrator to enter a key that would be used to calculate the HMAC hash, if the user/administrator was in the process of modifying an event log they could simply not enter the key until the Event Log Service had been restarted. Perhaps some form of peer based system that allows the servers to remotely monitor each other would be able to prevent this. Either that or base the HMAC key on a digital certificate.

A third area that would require more work is with how the server stores the log files. Currently there is no method in place for being able to check if events have been added or removed while they were off-site. This could have be addressed by allowing the client to build a local log file that is identical to that on the server. At a given interval a hash could be produced on both the client and the server, and then exchanged with one another. Any discrepancies would then be highlighted.

It would also be good if the system could be tested in a ‘real life’ environment, to see how it copes with multiple users accessing files at the same time. According to the tests that have been carried out, in theory, it should work fine, as the event logger application is monitoring the file system not the users.

8 References

Alles, MG. Kogan, A. and Vasarhelyi, MA (2004) *Restoring auditor credability: tertiary monitoring and logging of continuous assurance systems.* Elsevier Inc.

Armour Forensics (2005) *Master File Table Defined URL:* http://www.forensics-intl.com/def11.html [Accessed 29th November 2007]

Baryamureeba, V. and Tushabe, F. (2004) *The Enhanced Digital Investigation Process Model* Digital Forensic Research Workshop

Brown, CLT. (2005). *Computer Evidence Collection and Preservation* Charles River Media

Carrier, B. and Spafford E. (2004) An Event-Based Digital Forensic Investigation Framework. Purdue University

CodeProject. (2008). *Public Key RSA Encryption in C# .NET.* Accessed 7th 5 2008 from URL: http://www.codeproject.com/KB/security/RSACryptoPad.aspx

Cohen, F. (2006) *Challenges to Digital Forensic Evidence* Fred Cohen and Associates

Forte, D and Power, R. (2007) *A Tour Through The Realms of Anti-Forensics* Elsevier Inc.

Forte, D. (2005). *Log management for effective incident response* Network Security.

Forte, D. (2004). *The 'Art' of Log File Correlation* Elsevier Ltd

Gladyshev (2004) *Formalising Event Reconstruction in Digital Investigations* http://www.gladyshev.info/publications/thesis/chapter3.pdf [Accessed 27th November 2007]

Harris, Ryan (2006). *Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem* Purdue University

iFrame. (2008). *Hash Calculator*. Accessed 7th 5 2008 from URL: http://iframe.in/hash-calculator/

Kent, K. and Souppaya, M. (2006). *Guide to Computer Security Log Management* National Institute of Standards and Technology

Kessler (2004). *An Overview of Steganography for the Computer Forensics Examiner* Forensic Science Communications

Liu, V. (2005). *Metasploit antiforensics project* Metaspolit

Microsoft Corporation (2007). *Net start*. Retrieved November 26, 2007 from URL: http://technet2.microsoft.com/WindowsServer/en/library/6c0a1e0b-ea5d-4a38-9682-944d17cfe11c1033.mspx

Microsoft Corporation (2008). FileSystemWatcher Class. Accessed 21st April 2008. URL: http://msdn2.microsoft.com/en-us/library/system.io.filesystemwatcher.aspx

Murphy, J. (2006). *Forensic Readiness* Dexisive Inc

Nair, SK. Gamage, C. Dasti, MT. Crispo, B and Tanenbaum, AS (2006). *Countering Digital Forensics: An Identity Based Ephemerizer Cryptosystem* Vrije Universiteit

NIST (2006). *Forensic Techniques: Helping organisations improve their responses to information security incidents.* National Institute of Standards and Technology

NIST (2006). *Log Management: Using computer and network records to improve information security.* National Institute of Standards and Technology

OED. (2008). *Dictionary Facts*. Retrieved 7th May 2008 from URL: http://www.oed.com/about/facts.html

Palmer, G. (2001). *A Road Map for Digital Forensics Research* Digital Forensic Research Workshop

Panigrahi, P. (2006). *Discovering Fraud in Forensic Accounting Using Data Mining Techniques* The Chartered Accountant

Postnote (2006) *Computer Crime* Parliamentary Office of Science and Technology. Issue 271

Rogers, M. (2005). *Hard Challenges for Digital Forensics* http://www.mser.gov.bc.ca/privacyaccess/Conferences/Feb2005/ConfPresentations/Marcus_Rogers.pdf [accessed 26th November 2007]

Westphal, K (2001) *Secure Remote Log Servers Using SCP* http://www.securityfocus.com/infocus/1394 [accessed 27th November 2007]

Wikipedia (2008). Unix Time. Accessed 21st April 2008. URL: http://en.wikipedia.org/wiki/Unix_time

9 Appendices

Appendix A: Diary Sheets

Appendix B: Preliminary Gantt Chart

Appendix C: Client Code

Appendix D: Server Code

Appendix E: Event Viewer Code

Appendix F: Tester Application

Appendix G: Processor Monitor

Appendix H: HMAC Brute Force Cracker

Appendix I: Windows Event Log Tests

Appendix A:

Diary Sheets

NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Barrie Codona **Supervisor:** Bill Buchanan

Date: 26th October 2007 **Last diary date:** 19th October 2007

Objectives:

1. Investigate previous court cases that have made use of the event log.
2. Investigate 'event numbers' to further understand what they relate to.
3. Investigate the log to try and define start and end points for each entry, and then translate the hex data to what is displayed to the user.
4. Attempt to replace the event log with one supplied from a donor pc.
5. Modify the 'client logon id' to try and change the computer name and user name.
6. Modify the 32 bit hex date and note the results.

Progress:

1. Tracked down the transcript from the Dr Harold Shipman case and a reference that describes what he did and why he did it.
2. Found various sources describing the Event ID's
3. Work In Progress
4. Successfully managed to copy the security log from the donor pc. This produced some interesting results.
5. Altering the Client Logon ID has no effect on the Username that was used when the record was created, more investigation needs to be done here.
6. Successfully managed to alter the 32 bit time & date stamp in the security log.

Supervisor's Comments:

NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Barrie Codona **Supervisor:** Bill Buchanan

Date: 2nd November 2007 **Last diary date:** 26th October 2007

Objectives:

1. Further investigation into how the event log stores the time & date.
2. Automate the function of copying a 'modified' log file.
3. Further investigation into Dr Harold Shipman.
4. Begin investigation into current and previous research (literature review).

Progress:

1. It was discovered that the time is a count of the number of seconds that have passed since 00:00:00 01/01/1970
2. A console application has been developed that will prevent the event service from starting after the computer has been reset, thus allowing the log file to be replaced. More work is required to try and restart the service automatically.
3. Greater details of the actions of Dr Shipman have been discovered; this includes the database application that he was using (MicroDoc), and also, the company that did the digital forensic investigation (Vogon).
4. Work In Progress

Supervisor's Comments:

NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Barrie Codona **Supervisor:** Bill Buchanan

Date: 9th November 2007 **Last diary date:** 2nd November 2007

Objectives:

1. Investigate the structure of NTFS to allow the by-passing of the operating system
2. Investigate the possibility of modifying the contents of the hard disc while the event service is still running.
3. Investigate the possibilities of modifying the contents of the MFT. (Getting it to point to another file).
4. Begin development of a system that will create a real time back up of the event logs.

Progress:

1. It was discovered that NTFS uses a system called the Master File System (MFT). This is contained on the root directory of the boot drive, its file name is $MFT. The MFT is a relation database that contains various information about all the files on the drive.
2. Using 'Directory Snoop' to examine the MFT. This provides some information on the sectors of the event log files. Unfortunately this test did not return the results that were hoped for, however it has given some insight on how the event service works.
3. Based upon the results from the last test, it has been concluded that even if it were possible to modify the MFT and get it to point to a new file, the system would overwrite the contents of the file with the contents it has in memory when the computer was shutdown. Perhaps the memory could be modified in a similar way to the disc.
4. Contained within the Week 6 Weekly Report is an initial design specification of what the application might do.

Supervisor's Comments:

NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Barrie Codona **Supervisor:** Bill Buchanan

Date: 16th November 2007 **Last diary date:** 9th November 2007

Objectives:

1. Make contact with Jamie Graves and ask for assistance finding Digital Forensic Journals.
2. Search Digital Forensic web sites for relevant journals.
3. Read through the journals and take notes.
4. Start a draft version of the Literature Review.

Progress:

1. Emailed Jamie and received a reply. Jamie suggested a few sites that he thought maybe of relevance, a couple of these sites contained a large amount of relevant information. Patitularly found the following web sites the most helpful:

   www.sciencedirect.com

   www.acm.org

   Still trying to perfect the Google search strings, but this tip has proven use for tracking down articles that the other web sites don't give free access too.
2. So far managed to download approx 46 documents that have some relevance to the project.
3. Currently have read through 17 of the journals and managed to exclude 7 of them as not being directly related to the project.
4. This weeks report is the start of the draft Literature Review.

Supervisor's Comments:

NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Barrie Codona **Supervisor:** Bill Buchanan

Date: 30th November 2007 **Last diary date:** 16th November 2007

Objectives:

1. Continue to search Digital Forensic web sites for relevant journals.
2. Start work on Literature Review.
3. Design a structure for the report.
4. Complete the report.

Progress:

1. So far managed to download approx 78 documents that have some relevance to the project.
2. Began working on my Literature Review.
3. The report is built around the following topics:
    a. Introduction
    b. Digital Forensics
    c. Anti Forensics
    d. Log Management
    e. Conclusion
    f. References
4. The report is now completed and (not including references) spans 7 pages and contains 2,625 words.

Supervisor's Comments:

NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Barrie Codona **Supervisor:** Bill Buchanan

Date: 7th December 2007 **Last diary date:** 30th November 2007

Objectives:

1. Create a Gantt Chart for the project
2. Modify the picture in the Literature Review using MS Visio
3. Create an introduction for the project.

Progress:

1. Based upon previous projects that are available from Bill Buchanan's web site and from the feedback from last weeks meeting an out line for the overall report was produced, this allowed a Gantt Chart to be developed that showed the timeline for the project, giving various elements of the report and their milestones. These included:
    a. Introduction
    b. Theory
    c. Literature Review
    d. Design
    e. Implementation
    f. Evaluation
    g. Conclusions
    h. References
    i. Appendices
2. The picture displaying the 'Log architecture with time stamping appliance' was recreated using Visio and inserted into the Literature Review report
3. The introduction to the project has been started, but at the moment is incomplete.

Supervisor's Comments:

NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Barrie Codona **Supervisor:** Bill Buchanan

Date: 14th December 2007 **Last diary date:** 7th December 2007

Objectives:

1. Continue work on an introduction for the project.

Progress:

1. Work has continued on the Introduction for the project, it is almost complete. It starts of with the initial sentence that was supplied in the project hand book "The windows event log is used in digital forensic cases…" and then gives a couple of examples of where it has been used in digital forensic cases. It then introduces the reader to some of the security vulnerabilities of the event logging service. The report then provides some background information about the event log and that, according to Microsoft; it was designed to be used as a diagnostic tool. An example of how a logging server would be setup in a corporate environment is given; this involves having the log server connected directly, via a firewall, to a domain controller using a second NIC in the server. The report then introduces the design flaws that are in the event logging service and how more accurate information, with regards to the current state of the system, needs to be captured.

Supervisor's Comments:

NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Barrie Codona **Supervisor:** Bill Buchanan

Date: 20th February 2008 **Last diary date:** 14th December 2007

Objectives:

1. Begin writing the audit log software.
2. Separate the application into 3 separate agents, one for capturing the events, one for storing the events and one for viewing the events.

Progress:

1. Initially an event driven Windows form was created using the dot net framework, it specifically used the 'file system watcher' to monitor and report back any modifications that have been made to any files on the local drive.
2. The first two agents are nearing completion, currently the capture agent is event driven and captures any modifications that take place of the local file system and the user that caused these events, it then formats this information into a string and generates an HMAC checksum which is tagged onto the end of the string, using a TCP connection this is then sent to the logging agent. The logging agent, which is also event driven, stamps the received message with its local date & time and writes these events sequentially to a file.

Supervisor's Comments:

NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Barrie Codona **Supervisor:** Bill Buchanan

Date: 19th March 2008 **Last diary date:** 20th February 2008

Objectives:

1. Convert all the event tags to an XML format.
2. Encrypt the client server communication using RSA.
3. Ensure that both the client and server applications are not missing any events that happen.

Progress:

1. A fairly trivial modification to the application, but one that will allow the outputted data to be more easily read by other applications. This is not formatted in true XML but it does use the opening and closing tags for each string of information. For example <clientTime>????</clientTime>
2. Using the supplied sample code provided by Bill's Advanced Security and Forensic Computing module, a test application was developed that would allow for the program to be reconstructed from a windows environment to a console one. It was also modified so that the public and private keys were automatically generated when a client connects to the server. The public key is then sent to the server and allows for secure communication between them both. This worked fine for small amounts of information being sent, but larger amounts caused the RSA function to crash, this was probably because the key being used was smaller than the message that is being sent – it was resolved by breaking the original message into smaller sections and then encrypting each section and rebuilding the original message at the server. The next problem encountered was an "Invalid character" error in the decrypted string, after a lot of time investigating this it turned out to be the size of the TCP packets were too small and some information was being dropped.
3. A very simple application was developed that uses a 'for loop' to generate 1000 text files that are all sequentially numbered from 0 to 999. The event monitoring software is started up and its logs are manually checked to ensure that it has captured all of these events. Initially it failed to log all of the events but this was easily sorted by increasing the buffer size that the 'FileSystemWatcher' uses.

Supervisor's Comments:

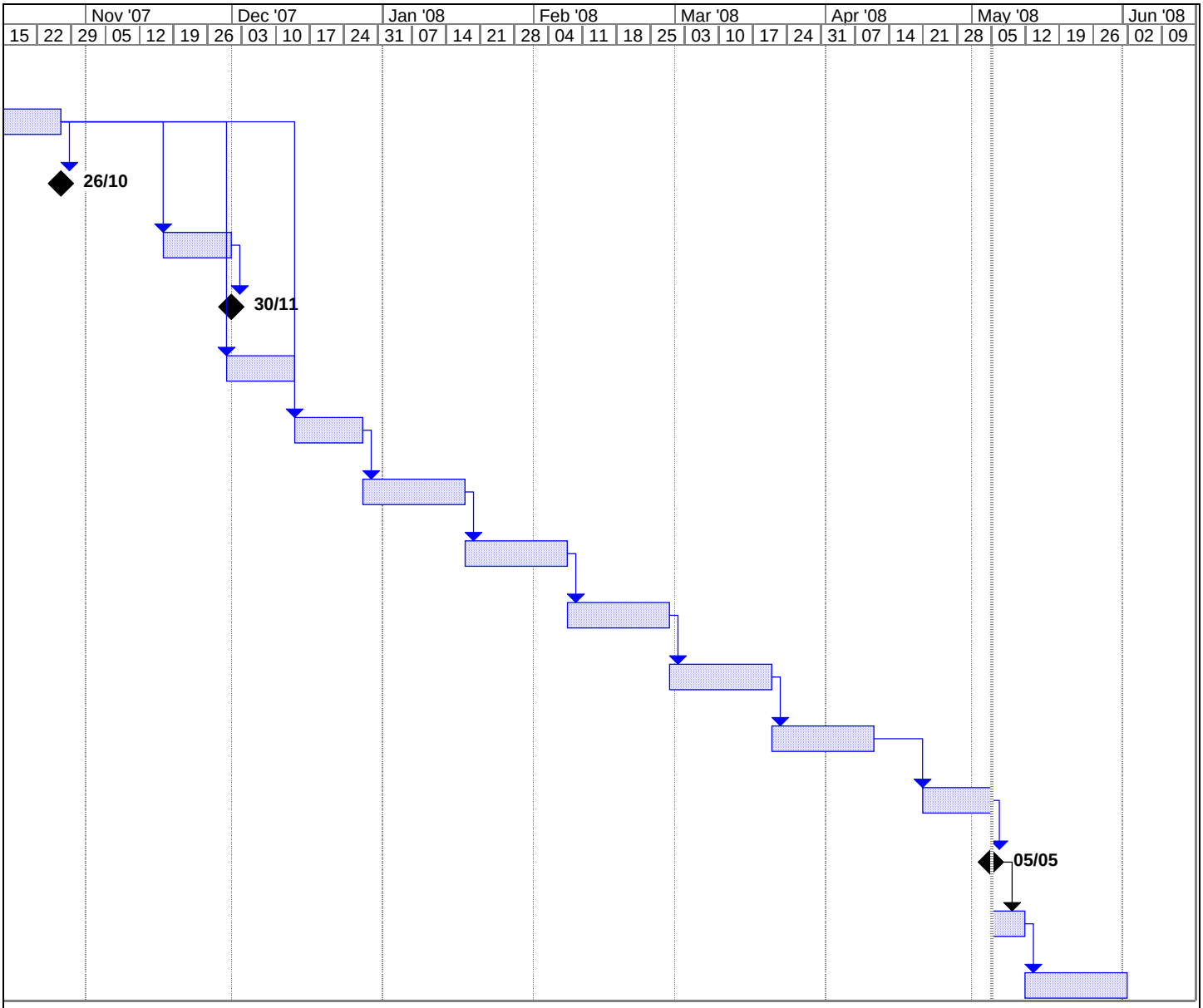
Appendix B:

Preliminary Gantt Chart

ID	Task Name	Duration	Start	Finish	Predecessors
1					
2					
3	Initial Project Overview	14 days	Sat 13/10/07	Fri 26/10/07	
4					
5	Submit Initial Project Overview	0 days	Fri 26/10/07	Fri 26/10/07	3
6					
7	Literature Review	14 days	Sat 17/11/07	Fri 30/11/07	3
8					
9	Submit Literature Review	0 days	Fri 30/11/07	Fri 30/11/07	7
10					
11	Introduction	14 days	Fri 30/11/07	Thu 13/12/07	3
12					
13	Theory	14 days	Fri 14/12/07	Thu 27/12/07	3
14					
15	Design	21 days	Fri 28/12/07	Thu 17/01/08	13
16					
17	Implementation	21 days	Fri 18/01/08	Thu 07/02/08	15
18					
19	Evaluation	21 days	Fri 08/02/08	Thu 28/02/08	17
20					
21	Conclusions	21 days	Fri 29/02/08	Thu 20/03/08	19
22					
23	References & Apendices	21 days	Fri 21/03/08	Thu 10/04/08	21
24					
25	Create Poster	14 days	Mon 21/04/08	Sun 04/05/08	23
26					
27	Submit report to Turnitin	0 days	Mon 05/05/08	Mon 05/05/08	25
28					
29	Attend Poster Session	7 days	Mon 05/05/08	Sun 11/05/08	27
30					
31	Viva Exam	21 days	Mon 12/05/08	Sun 01/06/08	29

Project: Project1.mpp
Date: Mon 05/05/08

Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline



Project: Project1.mpp
Date: Mon 05/05/08

Task		Project Summary	
Split		External Tasks	
Progress		External Milestone	
Milestone		Deadline	
Summary			

Appendix C:

Client Application Code

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.IO;
using System.Diagnostics;
using System.Security.Cryptography;
using System.Text.RegularExpressions;
using System.Security.Principal;

public class SimpleTcpClient
{
    public static void Main()
    {
        GlobalClass.myCount = 0;
        GlobalClass.user = WindowsIdentity.GetCurrent().Name;
        Console.WriteLine("Enter a key");
        GlobalClass.key = Convert.ToString(Console.ReadLine());

        // Generate AES Keys

        AES.passPhrase = Generate(40, 60);  // Random Length between 40-60
        AES.saltValue = Generate(40, 60);   // Random Length between 40-60
        AES.hashAlgorithm = "SHA1";         // can be "MD5"
        AES.passwordIterations = 2;         // can be any number
        AES.initVector = Generate(16, 16);  // RFixed Length of 16
        AES.keySize = 256;                  // can be 192 or 128

        // Connect to server
        connecttoserver();

        // Create a new FileSystemWatcher and set its properties.
        FileSystemWatcher watcher = new FileSystemWatcher();
        watcher.Path = "c:\\Test\\";
        watcher.NotifyFilter = NotifyFilters.LastAccess | NotifyFilters.LastWrite
           | NotifyFilters.FileName | NotifyFilters.DirectoryName;
        watcher.Filter = "*.*";
        watcher.IncludeSubdirectories = true;

        // Add event handlers.
        watcher.Changed += new FileSystemEventHandler(OnChanged);
        watcher.Created += new FileSystemEventHandler(OnChanged);
        watcher.Deleted += new FileSystemEventHandler(OnChanged);
        watcher.Renamed += new RenamedEventHandler(OnRenamed);

        // Begin watching.
        watcher.InternalBufferSize = 131072; //128 KB
        watcher.EnableRaisingEvents = true;

        // Create 2nd FileSystemWatcher and set its properties.
        FileSystemWatcher watcher2 = new FileSystemWatcher();
        watcher2.Path = "c:\\Test 2\\";
        watcher2.NotifyFilter = NotifyFilters.LastAccess | NotifyFilters.LastWrite
           | NotifyFilters.FileName | NotifyFilters.DirectoryName;
        watcher.Filter = "*.*";
        watcher.IncludeSubdirectories = true;

        // Add event handlers.
        watcher2.Changed += new FileSystemEventHandler(OnChanged);
        watcher2.Created += new FileSystemEventHandler(OnChanged);
        watcher2.Deleted += new FileSystemEventHandler(OnChanged);
        watcher2.Renamed += new RenamedEventHandler(OnRenamed);

        // Begin watching.
        watcher2.EnableRaisingEvents = true;

        while (true)
        {
            Console.WriteLine("Enter message to send");
            string message = Convert.ToString(Console.ReadLine());
            if (message == "exit")
                break;
            sendmsg(message);
        }
```

```
        Console.WriteLine("Disconnecting from server...");
        GlobalClass.server.Shutdown(SocketShutdown.Both);
        GlobalClass.server.Close();
    }

    public static string ByteToString(byte[] buff)
    {
        string sbinary = "";
        for (int i = 0; i < buff.Length; i++)
        {
            sbinary += buff[i].ToString("X2"); // hex format
        }
        return (sbinary);
    }

    static void connecttoserver()
    {
        Console.Write("Connecting to server... ");
        GlobalClass.data = new byte[10240];
        IPHostEntry host = Dns.GetHostEntry("localhost");
        IPAddress ipAddr = host.AddressList[0];
        IPEndPoint ipep = new IPEndPoint(ipAddr, 13000);
        GlobalClass.server = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
    ProtocolType.Tcp);
        try
        {
            GlobalClass.server.Connect(ipep);
        }
        catch (SocketException e)
        {
            Console.WriteLine("Failed!");
            Console.WriteLine(e.ToString());
            Console.ReadLine();
            return;
        }
        Console.WriteLine("Done!");
        GlobalClass.recv = GlobalClass.server.Receive(GlobalClass.data);
        GlobalClass.stringData = Encoding.ASCII.GetString(GlobalClass.data, 0, GlobalClass.
    recv);
        Console.WriteLine(GlobalClass.stringData);

        //receive public key
        GlobalClass.recv = GlobalClass.server.Receive(GlobalClass.data);
        GlobalClass.justPublicKey = Encoding.ASCII.GetString(GlobalClass.data, 0,
    GlobalClass.recv);
        Console.WriteLine(GlobalClass.justPublicKey);

        // Convert Shared Key data to XML string.
        string secretKey = "<?xml version=\"1.0\"?><AES>";
        secretKey += "<passPhrase>" + AES.passPhrase + "</passPhrase>";
        secretKey += "<saltValue>" + AES.saltValue + "</saltValue>";
        secretKey += "<hashAlgorithm>" + AES.hashAlgorithm + "</hashAlgorithm>";
        secretKey += "<passwordIterations>" + Convert.ToString(AES.passwordIterations) + "
    </passwordIterations>";
        secretKey += "<initVector>" + AES.initVector + "</initVector>";
        secretKey += "<keySize>" + Convert.ToString(AES.keySize) + "</keySize></AES>";

        // Encrypt Shared Key With RSA

        int dwKeySize = 1024;
        RSACryptoServiceProvider RSAProvider = new RSACryptoServiceProvider(dwKeySize);
        RSAProvider.FromXmlString(GlobalClass.justPublicKey);
        int keySize = dwKeySize / 8;
        byte[] bytes = Encoding.UTF32.GetBytes(secretKey);
        int maxLength = keySize - 42;
        int dataLength = bytes.Length;
        int iterations = dataLength / maxLength;
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i <= iterations; i++)
        {
            byte[] tempBytes = new byte[(dataLength - maxLength * i > maxLength) ?
    maxLength : dataLength - maxLength * i];
            Buffer.BlockCopy(bytes, maxLength * i, tempBytes, 0, tempBytes.Length);
            byte[] encryptedBytes = RSAProvider.Encrypt(tempBytes, true);
```

```
            stringBuilder.Append(Convert.ToBase64String(encryptedBytes));
        }
        string encryptedKey = Convert.ToString(stringBuilder);

        // Send Shared Key to Server.
        GlobalClass.server.Send(Encoding.ASCII.GetBytes(encryptedKey));
    }

    static void sendmsg(string chkMessage)
    {
        System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();
        byte[] keyByte = encoding.GetBytes(GlobalClass.key);
        HMACSHA1 hmac = new HMACSHA1(keyByte);
        byte[] messageBytes = encoding.GetBytes(chkMessage);
        byte[] hashmessage = hmac.ComputeHash(messageBytes);

        chkMessage = chkMessage + "<hmacCheck>" + ByteToString(hashmessage) + "</hmacCheck>↙
    ";

        string cipherText = Encrypt(chkMessage, AES.passPhrase, AES.saltValue, AES.        ↙
    hashAlgorithm, AES.passwordIterations, AES.initVector, AES.keySize);

        GlobalClass.server.Send(Encoding.ASCII.GetBytes(cipherText));
        GlobalClass.data = new byte[10240];
        GlobalClass.recv = GlobalClass.server.Receive(GlobalClass.data);
        GlobalClass.stringData = Encoding.ASCII.GetString(GlobalClass.data, 0, GlobalClass.↙
    recv);
        GlobalClass.myCount++;
        Console.Write(GlobalClass.myCount + " ");
    }
    public static string Encrypt(string plainText,string passPhrase,string saltValue,string↙
     hashAlgorithm,int passwordIterations,string initVector,int keySize)
    {
        byte[] initVectorBytes = Encoding.ASCII.GetBytes(initVector);
        byte[] saltValueBytes = Encoding.ASCII.GetBytes(saltValue);
        byte[] plainTextBytes = Encoding.UTF8.GetBytes(plainText);
        PasswordDeriveBytes password = new PasswordDeriveBytes(passPhrase,saltValueBytes,   ↙
    hashAlgorithm,passwordIterations);
        byte[] keyBytes = password.GetBytes(keySize / 8);
        RijndaelManaged symmetricKey = new RijndaelManaged();
        symmetricKey.Mode = CipherMode.CBC;
        ICryptoTransform encryptor = symmetricKey.CreateEncryptor(keyBytes,initVectorBytes)↙
    ;
        MemoryStream memoryStream = new MemoryStream();
        CryptoStream cryptoStream = new CryptoStream(memoryStream,encryptor,              ↙
    CryptoStreamMode.Write);
        cryptoStream.Write(plainTextBytes, 0, plainTextBytes.Length);
        cryptoStream.FlushFinalBlock();
        byte[] cipherTextBytes = memoryStream.ToArray();
        memoryStream.Close();
        cryptoStream.Close();
        string cipherText = Convert.ToBase64String(cipherTextBytes);
        return cipherText;
    }

    public static void OnChanged(object source, FileSystemEventArgs e)
    {
        string strFileExt = getFileExt(e.FullPath);
        if (Regex.IsMatch(strFileExt, @"\.log", RegexOptions.IgnoreCase))
        {
            //ignore them
        }
        else
        {
            // Specify what is done when a file is changed, created, or deleted.
            string message = "<clientTime>" + DateTime.Now + "</clientTime><user>" +       ↙
    GlobalClass.user + "</user><changeType>" + e.ChangeType + "</changeType><fullPath>" + e↙
    .FullPath + "</fullPath>";
            sendmsg(message);
        }
    }

    public static void OnRenamed(object source, RenamedEventArgs e)
    {
```

```
        // Specify what is done when a file is renamed.
        string message = "<clientTime>" + DateTime.Now + "</clientTime><user>" +
    GlobalClass.user + "</user><changeType>" + e.ChangeType + "</changeType><fullPath>" + e
    .FullPath + "</fullPath>";
        sendmsg(message);
    }

    private static string getFileExt(string filePath)
    {
        if (filePath == null) return "";
        if (filePath.Length == 0) return "";
        if (filePath.LastIndexOf(".") == -1) return "";
        return filePath.Substring(filePath.LastIndexOf("."));
    }

    static class AES
    {
        private static string m_passPhrase;

        public static string passPhrase
        {
            get { return m_passPhrase; }
            set { m_passPhrase = value; }
        }

        private static string m_saltValue;

        public static string saltValue
        {
            get { return m_saltValue; }
            set { m_saltValue = value; }
        }

        private static string m_hashAlgorithm;

        public static string hashAlgorithm
        {
            get { return m_hashAlgorithm; }
            set { m_hashAlgorithm = value; }
        }

        private static int m_passwordIterations;

        public static int passwordIterations
        {
            get { return m_passwordIterations; }
            set { m_passwordIterations = value; }
        }

        private static string m_initVector;

        public static string initVector
        {
            get { return m_initVector; }
            set { m_initVector = value; }
        }

        private static int m_keySize;

        public static int keySize
        {
            get { return m_keySize; }
            set { m_keySize = value; }
        }
    }

    static class GlobalClass
    {
        private static string m_key = "";

        public static string key
        {
            get { return m_key; }
            set { m_key = value; }
```

```
        }

        private static string m_stringData = "";

        public static string stringData
        {
            get { return m_stringData; }
            set { m_stringData = value; }
        }

        private static Socket m_server;

        public static Socket server
        {
            get { return m_server; }
            set { m_server = value; }
        }

        private static byte[] m_data;

        public static byte[] data
        {
            get { return m_data; }
            set { m_data = value; }
        }

        private static int m_recv;

        public static int recv
        {
            get { return m_recv; }
            set { m_recv = value; }
        }

        private static int m_myCount;

        public static int myCount
        {
            get { return m_myCount; }
            set { m_myCount = value; }
        }

        private static string m_user;

        public static string user
        {
            get { return m_user; }
            set { m_user = value; }
        }
        private static string m_justPublicKey;

        public static string justPublicKey
        {
            get { return m_justPublicKey; }
            set { m_justPublicKey = value; }
        }
    }

    private static string PASSWORD_CHARS_LCASE = "abcdefgijkmnopqrstwxyz";
    private static string PASSWORD_CHARS_UCASE = "ABCDEFGHJKLMNPQRSTWXYZ";
    private static string PASSWORD_CHARS_NUMERIC = "0123456789";
    private static string PASSWORD_CHARS_SPECIAL = "*$-+?_=!%{}/";

    public static string Generate(int minLength, int maxLength)
    {
        if (minLength <= 0 || maxLength <= 0 || minLength > maxLength)
            return null;

        char[][] charGroups = new char[][]
        {
            PASSWORD_CHARS_LCASE.ToCharArray(),
            PASSWORD_CHARS_UCASE.ToCharArray(),
            PASSWORD_CHARS_NUMERIC.ToCharArray(),
            PASSWORD_CHARS_SPECIAL.ToCharArray()
```

```
        };

        int[] charsLeftInGroup = new int[charGroups.Length];
        for (int i = 0; i < charsLeftInGroup.Length; i++)
            charsLeftInGroup[i] = charGroups[i].Length;
        int[] leftGroupsOrder = new int[charGroups.Length];
        for (int i = 0; i < leftGroupsOrder.Length; i++)
            leftGroupsOrder[i] = i;
        byte[] randomBytes = new byte[4];
        RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
        rng.GetBytes(randomBytes);
        int seed = (randomBytes[0] & 0x7f) << 24 |
                    randomBytes[1] << 16 |
                    randomBytes[2] << 8 |
                    randomBytes[3];
        Random random = new Random(seed);
        char[] password = null;
        if (minLength < maxLength)
            password = new char[random.Next(minLength, maxLength + 1)];
        else
            password = new char[minLength];
        int nextCharIdx;
        int nextGroupIdx;
        int nextLeftGroupsOrderIdx;
        int lastCharIdx;
        int lastLeftGroupsOrderIdx = leftGroupsOrder.Length - 1;
        for (int i = 0; i < password.Length; i++)
        {
            if (lastLeftGroupsOrderIdx == 0)
                nextLeftGroupsOrderIdx = 0;
            else
                nextLeftGroupsOrderIdx = random.Next(0, lastLeftGroupsOrderIdx);
            nextGroupIdx = leftGroupsOrder[nextLeftGroupsOrderIdx];
            lastCharIdx = charsLeftInGroup[nextGroupIdx] - 1;
            if (lastCharIdx == 0)
                nextCharIdx = 0;
            else
                nextCharIdx = random.Next(0, lastCharIdx + 1);
            password[i] = charGroups[nextGroupIdx][nextCharIdx];
            if (lastCharIdx == 0)
                charsLeftInGroup[nextGroupIdx] = charGroups[nextGroupIdx].Length;
            else
            {
                if (lastCharIdx != nextCharIdx)
                {
                    char temp = charGroups[nextGroupIdx][lastCharIdx];
                    charGroups[nextGroupIdx][lastCharIdx] = charGroups[nextGroupIdx]
    [nextCharIdx];
                    charGroups[nextGroupIdx][nextCharIdx] = temp;
                }
                charsLeftInGroup[nextGroupIdx]--;
            }
            if (lastLeftGroupsOrderIdx == 0)
                lastLeftGroupsOrderIdx = leftGroupsOrder.Length - 1;
            else
            {
                if (lastLeftGroupsOrderIdx != nextLeftGroupsOrderIdx)
                {
                    int temp = leftGroupsOrder[lastLeftGroupsOrderIdx];
                    leftGroupsOrder[lastLeftGroupsOrderIdx] = leftGroupsOrder
    [nextLeftGroupsOrderIdx];
                    leftGroupsOrder[nextLeftGroupsOrderIdx] = temp;
                }
                lastLeftGroupsOrderIdx--;
            }
        }
        return new string(password);
    }
}
```

Appendix D:

Server Application Code

```
using System;
using System.Collections;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.IO;
using System.Security.Cryptography;
using System.Xml;


public class SimpleTcpSrvr
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[10240];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 13000);

        Socket newsock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
    ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Waiting for a client...");
        Socket client = newsock.Accept();
        IPEndPoint clientep = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Connected with {0} at port {1}", clientep.Address, clientep.
    Port);

        string welcome = "You are connected to the server.";
        data = Encoding.ASCII.GetBytes(welcome);
        client.Send(data, data.Length, SocketFlags.None);

        //Generate random keys
        Console.Write("Generating new key pairs... ");
        int dwKeySize = 1024;
        RSACryptoServiceProvider RSAProvider = new RSACryptoServiceProvider(dwKeySize);
        string publicAndPrivateKeys = RSAProvider.ToXmlString(true);
        string justPublicKey = RSAProvider.ToXmlString(false);
        Console.WriteLine("Done!");
        Console.Write("Sending public key to client... ");
        data = Encoding.ASCII.GetBytes(justPublicKey);
        client.Send(data, data.Length, SocketFlags.None);
        Console.WriteLine("Done!");

        // Receive Encrypted Shared Key from Client
        data = new byte[10240];
        recv = client.Receive(data);
        string encryptedKey = Encoding.ASCII.GetString(data, 0, recv);

        // Decrypt With Private Key

        RSAProvider.FromXmlString(publicAndPrivateKeys);
        int base64BlockSize = ((dwKeySize / 8) % 3 != 0) ? (((dwKeySize / 8) / 3) * 4) + 4
    : ((dwKeySize / 8) / 3) * 4;
        int iterations = encryptedKey.Length / base64BlockSize;
        ArrayList arrayList = new ArrayList();
        for (int i = 0; i < iterations; i++)
        {
            byte[] encryptedBytes = Convert.FromBase64String(encryptedKey.Substring
    (base64BlockSize * i, base64BlockSize));
            arrayList.AddRange(RSAProvider.Decrypt(encryptedBytes, true));
        }
        string decryptedKey = Encoding.UTF32.GetString(arrayList.ToArray(Type.GetType(
    "System.Byte")) as byte[]);

        // Assign values from XML string
        XmlDocument doc = new XmlDocument();
        doc.LoadXml(@decryptedKey);
        XmlNode element = doc.SelectSingleNode("/AES/passPhrase");
        string passPhrase = element.InnerText;
        element = doc.SelectSingleNode("/AES/saltValue");
        string saltValue = element.InnerText;
        element = doc.SelectSingleNode("/AES/hashAlgorithm");
        string hashAlgorithm = element.InnerText;
```

```
        element = doc.SelectSingleNode("/AES/passwordIterations");
        int passwordIterations = Convert.ToInt16(element.InnerText);
        element = doc.SelectSingleNode("/AES/initVector");
        string initVector = element.InnerText;
        element = doc.SelectSingleNode("/AES/keySize");
        int keySize = Convert.ToInt16(element.InnerText);
        if(!(Directory.Exists("C:\\Logs\\" + clientep.Address)))
            Directory.CreateDirectory("C:\\Logs\\" + clientep.Address);



        string logFile = "C:\\Logs\\" + clientep.Address +"\\EvtLog2.log";
        TextWriter tsw;
        try
        {
            tsw = File.AppendText(logFile);
        }
        catch
        {
            tsw = new StreamWriter(@logFile);
        }
        tsw.WriteLine("<Session><Connected>Connected with {0} at port {1}</Connected>",
    clientep.Address, clientep.Port);
        int myCount = 0;
        recv = 0;
        while (true)
        {
            data = new byte[10240];
            recv = client.Receive(data);
            if (recv == 0)
                break;
            string encryptedMessage = Encoding.ASCII.GetString(data, 0, recv);
            string decryptedMessage = Decrypt(encryptedMessage, passPhrase, saltValue,
    hashAlgorithm, passwordIterations, initVector, keySize);

            decryptedMessage = "<ServerTime>" + DateTime.Now + "</ServerTime>" +
    decryptedMessage;
            myCount++;
            Console.Write(myCount + " ");
            tsw.WriteLine(decryptedMessage);
            client.Send(data, recv, SocketFlags.None);
        }
        Console.WriteLine("Disconnected from {0}", clientep.Address);
        tsw.WriteLine("<Disconnected>Disconnected from {0}</Disconnected></Session>",
    clientep.Address);
        tsw.Close();
        client.Close();
        newsock.Close();
    }
    static void writekey(string publickey)
    {
        StreamWriter fs = new StreamWriter("public.xml");
        fs.Write(publickey);
        fs.Close();
    }
    public static string Decrypt(string cipherText,string passPhrase,string saltValue,
    string hashAlgorithm,int passwordIterations,string initVector,int keySize)
    {
        byte[] initVectorBytes = Encoding.ASCII.GetBytes(initVector);
        byte[] saltValueBytes = Encoding.ASCII.GetBytes(saltValue);
        byte[] cipherTextBytes = Convert.FromBase64String(cipherText);
        PasswordDeriveBytes password = new PasswordDeriveBytes(passPhrase,saltValueBytes,
    hashAlgorithm,passwordIterations);
        byte[] keyBytes = password.GetBytes(keySize / 8);
        RijndaelManaged symmetricKey = new RijndaelManaged();
        symmetricKey.Mode = CipherMode.CBC;
        ICryptoTransform decryptor = symmetricKey.CreateDecryptor(keyBytes,initVectorBytes)
    ;
        MemoryStream memoryStream = new MemoryStream(cipherTextBytes);
        CryptoStream cryptoStream = new CryptoStream(memoryStream,decryptor,
    CryptoStreamMode.Read);
        byte[] plainTextBytes = new byte[cipherTextBytes.Length];
        String plainText;
        try
```

```
        {
            int decryptedByteCount = cryptoStream.Read(plainTextBytes, 0, plainTextBytes.
    Length);
            memoryStream.Close();
            cryptoStream.Close();
            plainText = Encoding.UTF8.GetString(plainTextBytes, 0, decryptedByteCount);
        }
        catch
        {
            plainText = "Wrong Encryption Key Used";
            memoryStream.Close();
        }
        return plainText;
    }

}
```

Appendix E:

Event Viewer Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Collections;
using System.Security.Cryptography;

namespace Event_Viewer
{
    public partial class Form1 : Form
    {
        ArrayList lines = new ArrayList();
        int arrayItem = 0;

        public Form1()
        {
            InitializeComponent();
            readfile();
            LoadRecord();
        }

        private void readfile()
        {
            StreamReader sr = new StreamReader("c:\\Logs\\127.0.0.1\\EvtLog2.log");
            string line;
            while ((line = sr.ReadLine()) != null)
            {
                if(line.Contains("<ServerTime>"))
                lines.Add(line);
            }
            sr.Close();
            msgTxt.Text = "done";
        }

        private void LoadRecord()
        {
            try
            {
                string string1 = Convert.ToString(lines[arrayItem]);

                eventNoLbl.Text = Convert.ToString(arrayItem + 1);

                string myString1 = "<ServerTime>";
                string myString2 = "</ServerTime>";
                serverTimeLbl.Text = string1.Substring((string1.IndexOf(myString1) +
    myString1.Length), (string1.IndexOf(myString2) - (string1.IndexOf(myString1) +
    myString1.Length)));

                string myString3 = "<clientTime>";
                string myString4 = "</clientTime>";
                clientTimeLbl.Text = string1.Substring((string1.IndexOf(myString3) +
    myString3.Length), (string1.IndexOf(myString4) - (string1.IndexOf(myString3) +
    myString3.Length)));

                string myString5 = "<user>";
                string myString6 = "</user>";
                userLbl.Text = string1.Substring((string1.IndexOf(myString5) + myString5.
    Length), (string1.IndexOf(myString6) - (string1.IndexOf(myString5) + myString5.
    Length)));

                string myString7 = "<changeType>";
                string myString8 = "</changeType>";
                modTypeLbl.Text = string1.Substring((string1.IndexOf(myString7) + myString7
    .Length), (string1.IndexOf(myString8) - (string1.IndexOf(myString7) + myString7.
    Length)));

                string myString9 = "<fullPath>";
                string myString10 = "</fullPath>";
                pathLbl.Text = string1.Substring((string1.IndexOf(myString9) + myString9.
    Length), (string1.IndexOf(myString10) - (string1.IndexOf(myString9) + myString9.
```

```
    Length)));

                string myString11 = "<hmacCheck>";
                string myString12 = "</hmacCheck>";
                hmacLbl.Text = string1.Substring((string1.IndexOf(myString11) + myString11.↙
    Length), (string1.IndexOf(myString12) - (string1.IndexOf(myString11) + myString11.     ↙
    Length)));
                hmacChecker();
            }
            catch
            {
                MessageBox.Show("No Events Available to Display!","Error");
            }
        }

        private void hmacChecker()
        {
            string string1 = Convert.ToString(lines[arrayItem]);
            string myString1 = "<clientTime>";
            string myString2 = "</fullPath>";
            string chkMessage = string1.Substring(string1.IndexOf(myString1), ((string1.   ↙
    IndexOf(myString2) + myString2.Length)- string1.IndexOf(myString1)));
            msgTxt.Text = chkMessage;

            System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();
            byte[] keyByte = encoding.GetBytes(hmacTxt.Text);
            HMACSHA1 hmac = new HMACSHA1(keyByte);
            byte[] messageBytes = encoding.GetBytes(chkMessage);
            byte[] hashmessage = hmac.ComputeHash(messageBytes);
            hmacCheckLbl.Text = ByteToString(hashmessage);
            if (hmacCheckLbl.Text == hmacLbl.Text)
                hmacCheckLbl.Text += " : Hmac Passed!";
            else
                hmacCheckLbl.Text += " : Hmac Failed!";
        }

        public static string ByteToString(byte[] buff)
        {
            string sbinary = "";
            for (int i = 0; i < buff.Length; i++)
            {
                sbinary += buff[i].ToString("X2"); // hex format
            }
            return (sbinary);
        }

        private void nextBtn_Click(object sender, EventArgs e)
        {
            arrayItem++;
            if (arrayItem > lines.Count - 1)
                arrayItem = 0;
            LoadRecord();
        }

        private void prevBtn_Click(object sender, EventArgs e)
        {
            arrayItem--;
            if (arrayItem < 0)
                arrayItem = lines.Count - 1;
            LoadRecord();
        }

    }
}
```

Appendix F:

Tester Application Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.IO;

namespace EventLogTester
{
    public partial class Form1 : Form
    {
        int fileCounter = 1000;

        public Form1()
        {
            InitializeComponent();

            textBox1.Text = "File Counter = " + Convert.ToString(fileCounter);
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Process.Start("C:\\Documents and Settings\\Barrie\\My Documents\\Visual Studio 2005\\Projects\\simpletcpclient2-aes\\simpletcpclient2-aes\\bin\\Debug\\simpletcpclient2-aes.exe");
            Process.Start("C:\\Documents and Settings\\Barrie\\My Documents\\Visual Studio 2005\\Projects\\simpletcpserver-aes\\simpletcpserver-aes\\bin\\Debug\\simpletcpserver-aes.exe");
        }

        private void button2_Click(object sender, EventArgs e)
        {
            textBox1.Text += "\r\nGenerating " + Convert.ToString(fileCounter) + " files...";
            for (int i = 0; i < fileCounter; i++)
            {
                string myFileName = "c:\\Test\\Sample" + Convert.ToString(i) + ".txt";
                TextWriter tsw;
                tsw = new StreamWriter(@myFileName);
                tsw.Close();
            }
            textBox1.Text += "\r\nDone!";

        }

        private void button4_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < fileCounter; i++)
            {
                string myFileName = "c:\\Test\\Sample" + Convert.ToString(i) + ".txt";
                TextWriter tsw;
                if (File.Exists(myFileName))
                {
                    tsw = File.AppendText(myFileName);
                    for (int j = 0; j < 1000; j++)
                    {
                        tsw.Write("This file has been modified.  ");
                    }
                    tsw.Close();
                }
            }
        }

        private void button5_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < fileCounter; i++)
            {
                string myFileName = "c:\\Test\\Sample" + Convert.ToString(i) + ".txt";
                string newFileName = "c:\\Test\\NewSample" + Convert.ToString(i) + ".txt";
                if(!(File.Exists(newFileName)))
                {
```

```
                    File.Move(myFileName, newFileName);
                }
            }
        }

        private void button3_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < fileCounter; i++)
            {
                string myFileName = "c:\\Test\\Sample" + Convert.ToString(i) + ".txt";
                string newFileName = "c:\\Test\\NewSample" + Convert.ToString(i) + ".txt";
                if (File.Exists(myFileName))
                {
                    File.Delete(myFileName);
                }
                if (File.Exists(newFileName))
                {
                    File.Delete(newFileName);
                }
            }
        }

        private void button10_Click(object sender, EventArgs e)
        {
            fileCounter = Convert.ToInt32(textBox2.Text);
            textBox1.Text += ("\r\nFile Counter = " + fileCounter);
            textBox1.Select(textBox1.Text.Length + 1, 2);
            textBox1.ScrollToCaret();
        }

        private void button6_Click(object sender, EventArgs e)
        {
            textBox1.Text += "\r\n";
            Random num = new Random();
            int myNum = Convert.ToInt16(num.Next(1, fileCounter));
            textBox1.Text += myNum;
            for (int i = 0; i < fileCounter; i++)
            {
                if (myNum==i)
                {
                    string myFileName2 = "c:\\Test 2\\SevEvent.txt";
                    TextWriter tsw2;
                    if (File.Exists(myFileName2))
                    {
                        tsw2 = File.AppendText(myFileName2);
                        tsw2.Write("\r\nThis file has been modified.  Number: " + myNum);
                        tsw2.Close();
                    }
                }
                string myFileName = "c:\\Test\\Sample" + Convert.ToString(i) + ".txt";
                TextWriter tsw;
                tsw = new StreamWriter(@myFileName);
                tsw.Close();
            }
            textBox1.Select(textBox1.Text.Length + 1, 2);
            textBox1.ScrollToCaret();
        }
    }
}
```

Appendix G:

Processor Monitor Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;

namespace WindowsApplication6
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            if (timer1.Enabled == false)
            {
                richTextBox1.Text = "";
                timer1.Enabled = true;
            }
            else
            {
                timer1.Enabled = false;
            }
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            PerformanceCounter cpuCounter = new PerformanceCounter("Processor", "%
    Processor Time", "_Total", true);
            cpuCounter.NextValue();
            System.Threading.Thread.Sleep(100);// 0.1 second wait
            string myText = cpuCounter.NextValue() + "%";
            string myTime = Convert.ToString(DateTime.Now);
            richTextBox1.Text += myTime + " : " + myText + "\r\n";
            richTextBox1.Select(richTextBox1.Text.Length + 1, 2);
            richTextBox1.ScrollToCaret();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            timer1.Enabled = false;
            string savedFile = "";
            saveFileDialog1.InitialDirectory = "C:";
            saveFileDialog1.Title = "Save Output to File!";
            saveFileDialog1.FileName = "Output.txt";
            saveFileDialog1.Filter = "Text Files|*.txt|All Files|*.*";
            if (saveFileDialog1.ShowDialog() != DialogResult.Cancel)
            {
                savedFile = saveFileDialog1.FileName;
                richTextBox1.SaveFile(savedFile, RichTextBoxStreamType.PlainText);
            }
        }
    }
}
```

Appendix H:

HMAC Brute Force Cracker Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Security.Cryptography;

namespace WindowsApplication7
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            //textBox4.Text = "b3fd686d2d595f31a8cf41da9610a8aeb9e5e8b3"; //Key = 1
            //textBox4.Text = "13f468463b446125a2f78ac27a24a1303b9b5db6"; //Key = z1
            //textBox4.Text = "1c95580fedf33cca6cb560dc55322ca46b1a97f1"; //Key = aa
            //textBox4.Text = "cf97aef064220b2dfffd20ff8946d9b370cca5f4"; //Key = t4B
            //textBox4.Text = "0c94515c15e5095b8a87a50ba0df3bf38ed05fe6"; //Key = test
            //textBox4.Text = "ea1cbddaaa10898fb076c93ac46debacac6c5d67"; //Key = aV2We1
            textBox4.Text = "f1e16dce22437e68581b664331cb2674ab2182fe"; //Key = zzzz
            //textBox2.Text = "<clientTime>30/04/2008 11:19:28</clientTime><user>PC3\\
    Barrie</user><changeType>Created</changeType><fullPath>c:\\Test\\Sample4591.txt</
    fullPath>";
            textBox2.Text = "test";
        }

        private void button1_Click(object sender, EventArgs e)
        {
            setTime();
            button1.Enabled = false;
            progressBar1.Maximum = 75;
            progressBar1.Minimum = 0;
            progressBar2.Maximum = 75;
            progressBar2.Minimum = 0;
            progressBar3.Maximum = 75;
            progressBar3.Minimum = 0;
            progressBar4.Maximum = 75;
            progressBar4.Minimum = 0;
            progressBar5.Maximum = 75;
            progressBar5.Minimum = 0;
            textBox3.Text = "";
            bool over = false;
            if (textBox4.Text.Equals(""))
            {
                textBox3.Text = "Field Empty";
                over = true;
            }
            char ch1, ch2, ch3, ch4, ch5, ch6;
            // Find 1 character key
            progressBar1.Value = 0;
            progressBar2.Value = 0;
            if (!over)
            {
                textBox3.Text += "Checking all 1 character keys... ";
                textBox3.Refresh();
                for (ch1 = '0'; ch1 <= 'z' && !over; ch1++)
                {
                    progressBar1.Value +=1;
                    string myCh = Convert.ToString(ch1);
                    string myNewKey = checker(myCh);
                    if (textBox4.Text == myNewKey.ToLower())
                    {
                        string myTime = calcTime();
                        textBox3.Text += "Key Found: " + myCh + "\r\nFinnished at: " +
    myTime;
                        over = true;
                        break;
                    }
                }
                if (!over)
                {
```

```
                textBox3.Text += "Done! ";
            }
        }
        // Find 2 character key
        progressBar1.Value = 0;
        progressBar2.Value = 0;
        if (!over)
        {
            textBox3.Text += "\r\nChecking all 2 character keys... ";
            textBox3.Refresh();
            for (ch1 = '0'; ch1 <= 'z' && !over; ch1++)
            {
                progressBar1.Value += 1;
                // Char 2
                for (ch2 = '0'; ch2 <= 'z' && !over; ch2++)
                {
                    string myCh = Convert.ToString(ch1) +
                                  Convert.ToString(ch2);
                    string myNewKey = checker(myCh);
                    if (textBox4.Text == myNewKey.ToLower())
                    {
                        string myTime = calcTime();
                        textBox3.Text += "Key Found: " + myCh + "\r\nFinnished at: " + ↙
myTime;
                        over = true;
                        break;
                    }
                }
            }
            if (!over)
            {
                textBox3.Text += "Done! ";
            }
        }
        // Find 3 character key
        progressBar1.Value = 0;
        progressBar2.Value = 0;
        progressBar3.Value = 0;
        if (!over)
        {
            textBox3.Text += "\r\nChecking all 3 character keys... ";
            textBox3.Refresh();
            for (ch1 = '0'; ch1 <= 'z' && !over; ch1++)
            {
                progressBar2.Value += 1;
                progressBar1.Value = 0;
                this.Refresh();
                // Char 2
                for (ch2 = '0'; ch2 <= 'z' && !over; ch2++)
                {
                    progressBar1.Value += 1;
                    // Char 3
                    for (ch3 = '0'; ch3 <= 'z' && !over; ch3++)
                    {
                        string myCh = Convert.ToString(ch1) +
                                      Convert.ToString(ch2) +
                                      Convert.ToString(ch3);
                        string myNewKey = checker(myCh);
                        if (textBox4.Text == myNewKey.ToLower())
                        {
                            string myTime = calcTime();
                            textBox3.Text += "Key Found: " + myCh + "\r\nFinnished at: ↙
" + myTime;
                            over = true;
                            break;
                        }
                    }
                }
            }
            if (!over)
            {
                textBox3.Text += "Done! ";
            }
        }
```

```
            // Find 4 character key
            progressBar1.Value = 0;
            progressBar2.Value = 0;
            progressBar3.Value = 0;
            if (!over)
            {
                textBox3.Text += "\r\nChecking all 4 character keys... ";
                textBox3.Refresh();
                for (ch1 = '0'; ch1 <= 'z' && !over; ch1++)
                {
                    progressBar3.Value += 1;
                    progressBar2.Value = 0;
                    // Char 2
                    for (ch2 = '0'; ch2 <= 'z' && !over; ch2++)
                    {
                        progressBar2.Value += 1;
                        progressBar1.Value = 0;
                        this.Refresh();
                        // Char 3
                        for (ch3 = '0'; ch3 <= 'z' && !over; ch3++)
                        {
                            progressBar1.Value += 1;
                            // Char 4
                            for (ch4 = '0'; ch4 <= 'z' && !over; ch4++)
                            {
                                string myCh = Convert.ToString(ch1) +
                                              Convert.ToString(ch2) +
                                              Convert.ToString(ch3) +
                                              Convert.ToString(ch4);
                                string myNewKey = checker(myCh);
                                if (textBox4.Text == myNewKey.ToLower())
                                {
                                    string myTime = calcTime();
                                    textBox3.Text += "Key Found: " + myCh + "\r\nFinnished ↙
    at: " + myTime;
                                    over = true;
                                    break;
                                }
                            }
                        }
                    }
                }
                if (!over)
                {
                    textBox3.Text += "Done! ";
                }
            }
            // Find 5 character key
            progressBar1.Value = 0;
            progressBar2.Value = 0;
            progressBar3.Value = 0;
            progressBar4.Value = 0;
            if (!over)
            {
                textBox3.Text += "\r\nChecking all 5 character keys... ";
                textBox3.Refresh();
                for (ch1 = '0'; ch1 <= 'z' && !over; ch1++)
                {
                    progressBar3.Value = 0;
                    progressBar4.Value += 1;
                    // Char 2
                    for (ch2 = '0'; ch2 <= 'z' && !over; ch2++)
                    {
                        progressBar2.Value = 0;
                        progressBar3.Value += 1;
                        // Char 3
                        for (ch3 = '0'; ch3 <= 'z' && !over; ch3++)
                        {
                            progressBar1.Value = 0;
                            progressBar2.Value += 1;
                            this.Refresh();
                            // Char 4
                            for (ch4 = '0'; ch4 <= 'z' && !over; ch4++)
                            {
```

```
                            progressBar1.Value += 1;
                            // Char 5
                            for (ch5 = '0'; ch5 <= 'z' && !over; ch5++)
                            {
                                string myCh = Convert.ToString(ch1) +
                                              Convert.ToString(ch2) +
                                              Convert.ToString(ch3) +
                                              Convert.ToString(ch4) +
                                              Convert.ToString(ch5);
                                string myNewKey = checker(myCh);
                                if (textBox4.Text == myNewKey.ToLower())
                                {
                                    string myTime = calcTime();
                                    textBox3.Text += "Key Found: " + myCh + "\r\
nFinnished at: " + myTime;
                                    over = true;
                                    break;
                                }
                            }
                        }
                    }
                }
            }
            if (!over)
            {
                textBox3.Text += "Done! ";
            }
        }
        // Find 6 character key
        progressBar1.Value = 0;
        progressBar2.Value = 0;
        progressBar3.Value = 0;
        progressBar4.Value = 0;
        progressBar5.Value = 0;
        if (!over)
        {
            textBox3.Text += "\r\nChecking all 6 character keys... ";
            textBox3.Refresh();
            for (ch1 = '0'; ch1 <= 'z' && !over; ch1++)
            {
                progressBar4.Value = 0;
                progressBar5.Value += 1;
                // Char 2
                for (ch2 = '0'; ch2 <= 'z' && !over; ch2++)
                {
                    progressBar3.Value = 0;
                    progressBar4.Value += 1;
                    // Char 3
                    for (ch3 = '0'; ch3 <= 'z' && !over; ch3++)
                    {
                        progressBar2.Value = 0;
                        progressBar3.Value += 1;
                        // Char 4
                        for (ch4 = '0'; ch4 <= 'z' && !over; ch4++)
                        {
                            progressBar1.Value = 0;
                            progressBar2.Value += 1;
                            this.Refresh();
                            // Char 5
                            for (ch5 = '0'; ch5 <= 'z' && !over; ch5++)
                            {
                                progressBar1.Value += 1;
                                // Char 6
                                for (ch6 = '0'; ch6 <= 'z' && !over; ch6++)
                                {
                                    string myCh = Convert.ToString(ch1) +
                                                  Convert.ToString(ch2) +
                                                  Convert.ToString(ch3) +
                                                  Convert.ToString(ch4) +
                                                  Convert.ToString(ch5) +
                                                  Convert.ToString(ch6);
                                    string myNewKey = checker(myCh);
                                    if (textBox4.Text == myNewKey.ToLower())
                                    {
```

```
                                        string myTime = calcTime();
                                        textBox3.Text += "Key Found: " + myCh + "\r\
    nFinnished at: " + myTime;
                                        over = true;
                                        break;
                                    }
                                }
                            }
                        }
                    }
                }
                if (!over)
                {
                    textBox3.Text += "Done! ";
                }
            }
            button1.Enabled = true;
        }
        private void setTime()
        {
            label4.Text = Convert.ToString(DateTime.Now);
            label4.Refresh();
        }
        private string calcTime()
        {
            string myString = Convert.ToString(DateTime.Now);
            return (myString);
        }
        public static string ByteToString(byte[] buff)
        {
            string sbinary = "";
            for (int i = 0; i < buff.Length; i++)
            {
                sbinary += buff[i].ToString("X2"); // hex format
            }
            return (sbinary);
        }
        private string checker(string key)
        {
            string chkMessage = textBox2.Text;
            System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();
            byte[] keyByte = encoding.GetBytes(key);
            HMACSHA1 hmac = new HMACSHA1(keyByte);
            byte[] messageBytes = encoding.GetBytes(chkMessage);
            byte[] hashmessage = hmac.ComputeHash(messageBytes);
            return (ByteToString(hashmessage));
        }
    }
}
```

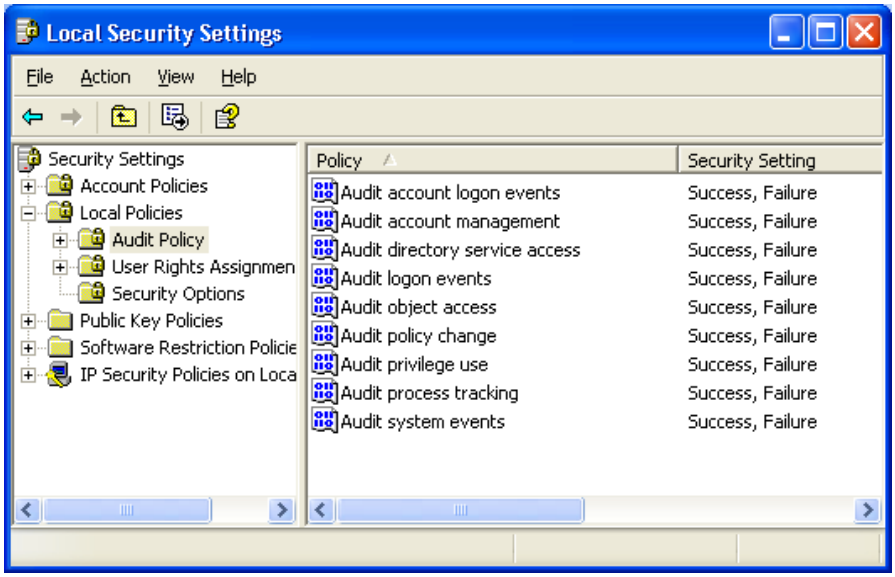
Appendix I:

Windows Event Log Tests

Project – Week 2

Setup security policy

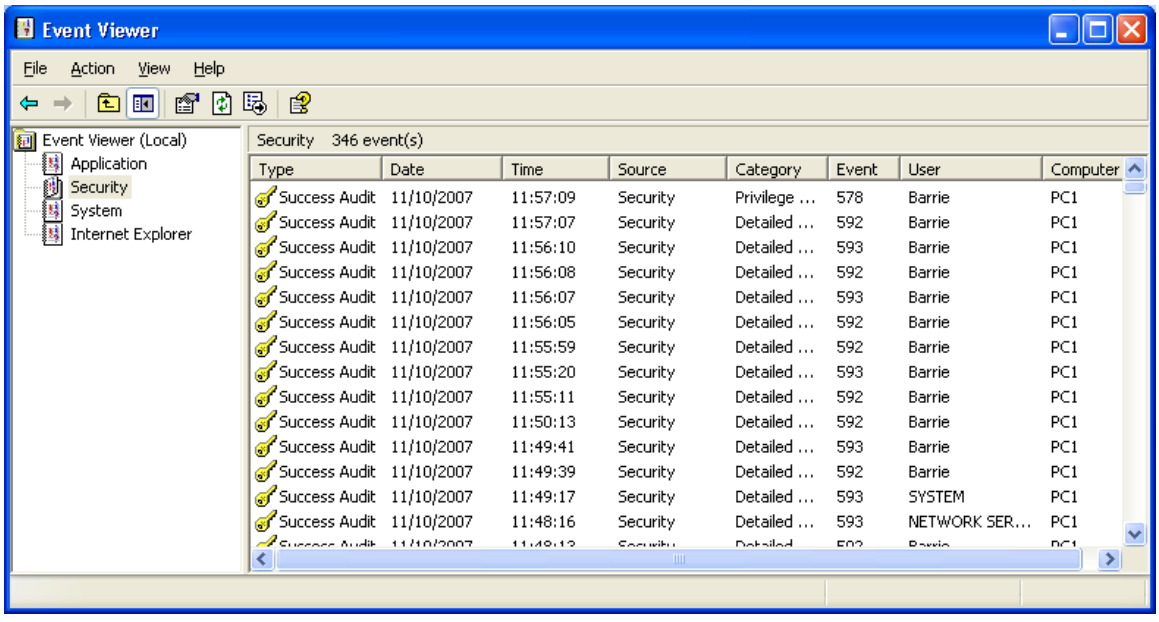
Control Panel > Admin Tools > Local Security Settings > Local Policy > Audit Policy



By default all of the policies were not set.

Display Security Events

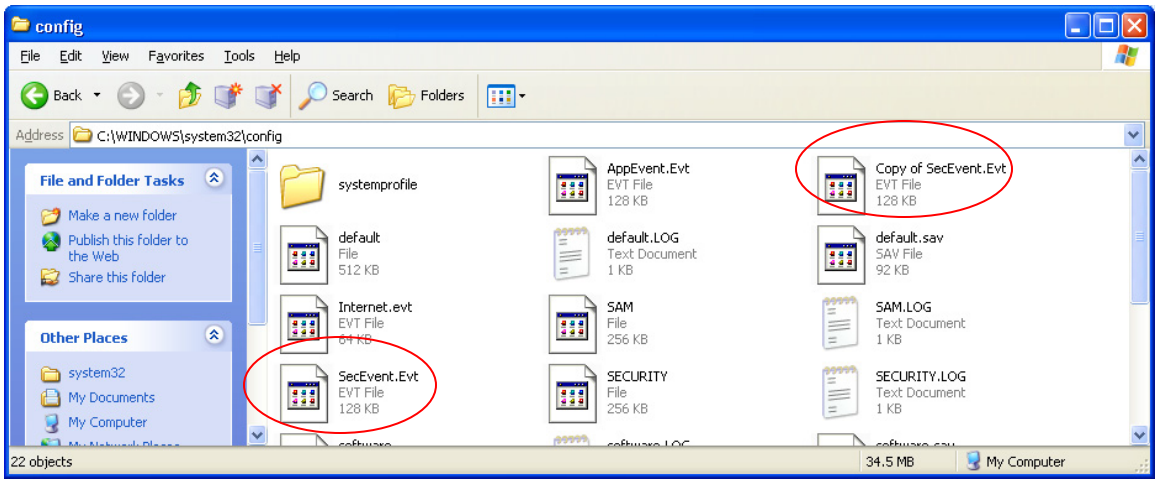
Control Panel > Admin Tools > Event Viewer > Security



Copy Security Log

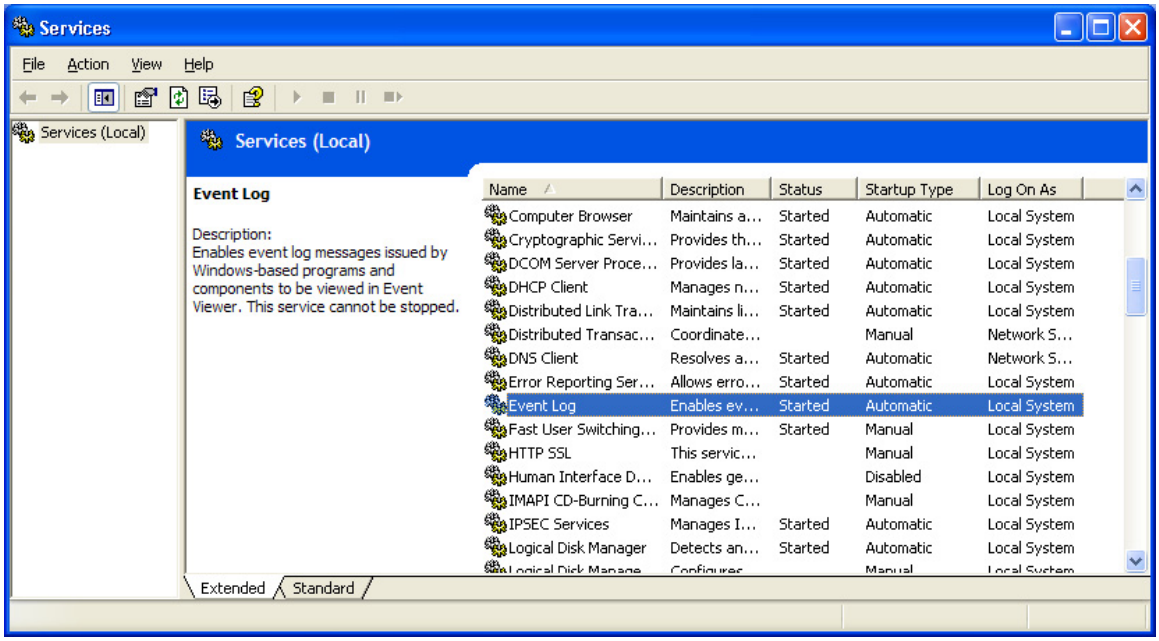
C:\windows\system32\config\SevEvent.Evt

This was copied and renamed to ‘Copy of SevEvent.Evt’



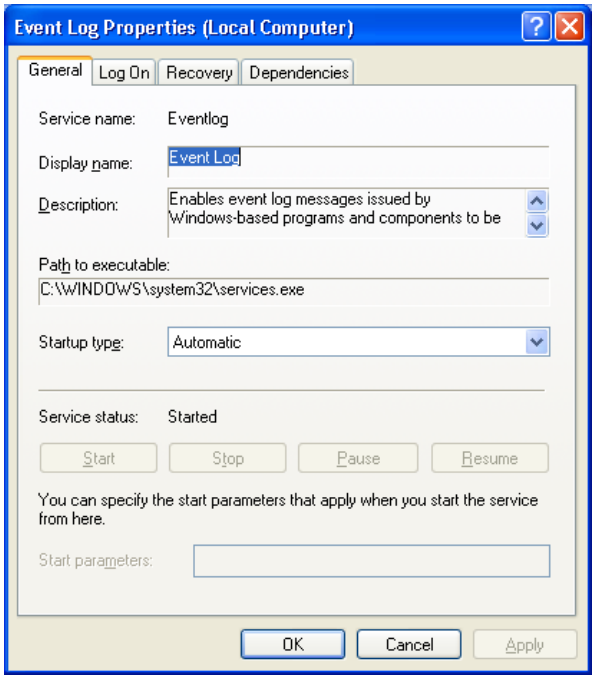
Stop Event Service

Control Panel > Admin Tools > Services > Event Log



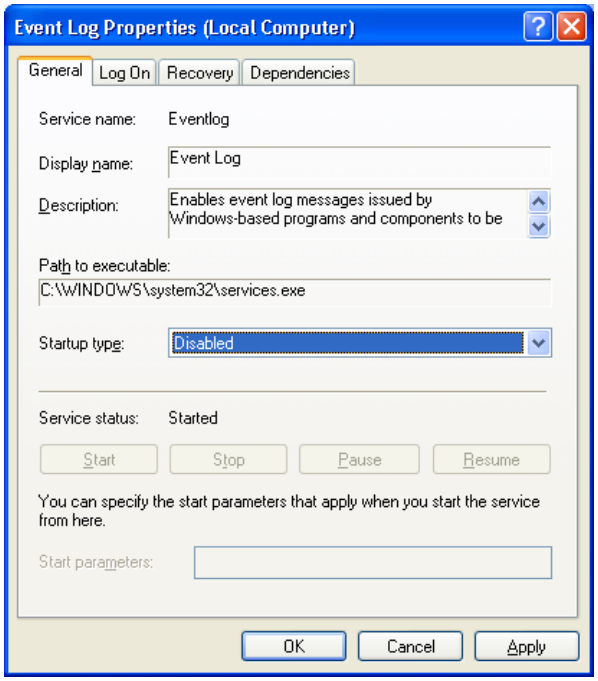
Right click and select ‘Properties’

The following dialog box is displayed:



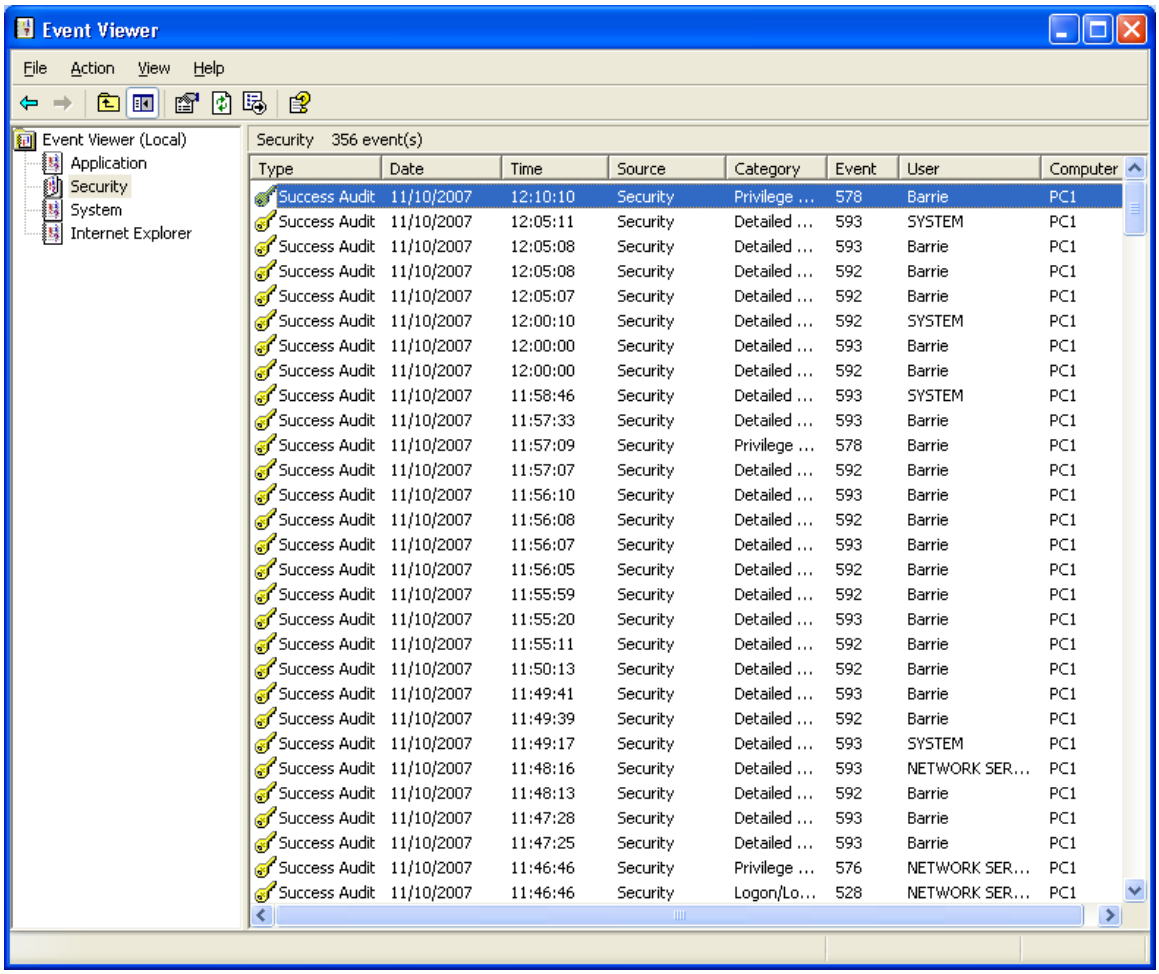
Note:

This service cannot be stopped after it has started, however it can be disabled from starting up the next time the computer is booted up.



Another look at the Security Log reviels that there have been some more events added since the last screen shot, it now contains 356 events.

This screen will be compared to later on.

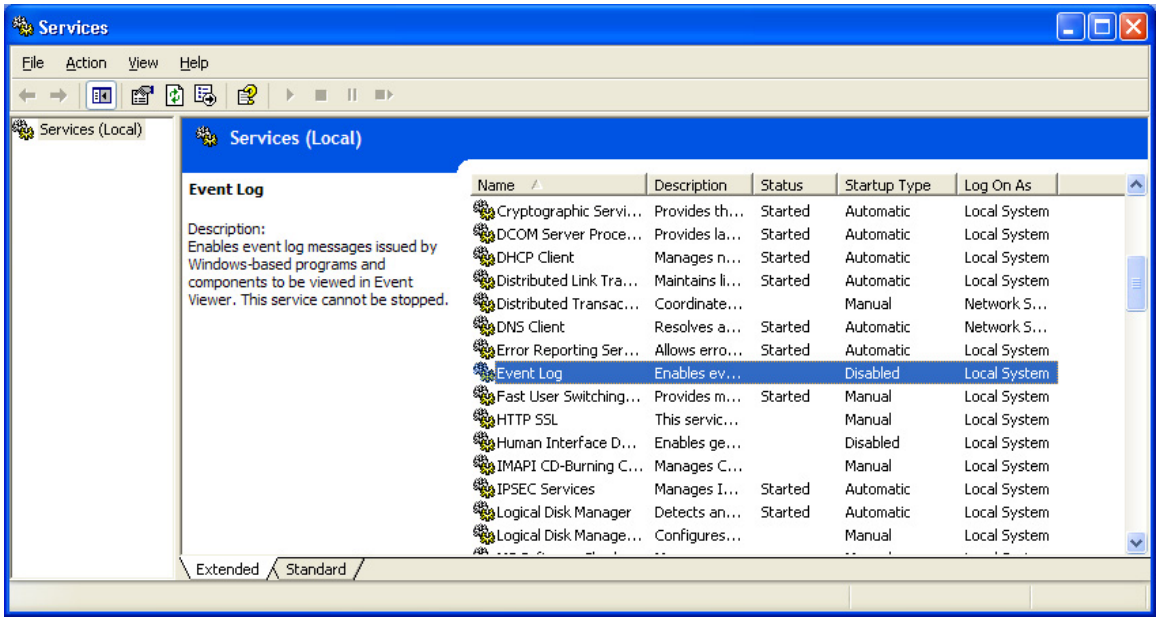


Type	Date	Time	Source	Category	Event	User	Computer
Success Audit	11/10/2007	12:10:10	Security	Privilege ...	578	Barrie	PC1
Success Audit	11/10/2007	12:05:11	Security	Detailed ...	593	SYSTEM	PC1
Success Audit	11/10/2007	12:05:08	Security	Detailed ...	593	Barrie	PC1
Success Audit	11/10/2007	12:05:08	Security	Detailed ...	592	Barrie	PC1
Success Audit	11/10/2007	12:05:07	Security	Detailed ...	592	Barrie	PC1
Success Audit	11/10/2007	12:00:10	Security	Detailed ...	592	SYSTEM	PC1
Success Audit	11/10/2007	12:00:00	Security	Detailed ...	593	Barrie	PC1
Success Audit	11/10/2007	12:00:00	Security	Detailed ...	592	Barrie	PC1
Success Audit	11/10/2007	11:58:46	Security	Detailed ...	593	SYSTEM	PC1
Success Audit	11/10/2007	11:57:33	Security	Detailed ...	593	Barrie	PC1
Success Audit	11/10/2007	11:57:09	Security	Privilege ...	578	Barrie	PC1
Success Audit	11/10/2007	11:57:07	Security	Detailed ...	592	Barrie	PC1
Success Audit	11/10/2007	11:56:10	Security	Detailed ...	593	Barrie	PC1
Success Audit	11/10/2007	11:56:08	Security	Detailed ...	592	Barrie	PC1
Success Audit	11/10/2007	11:56:07	Security	Detailed ...	593	Barrie	PC1
Success Audit	11/10/2007	11:56:05	Security	Detailed ...	592	Barrie	PC1
Success Audit	11/10/2007	11:55:59	Security	Detailed ...	592	Barrie	PC1
Success Audit	11/10/2007	11:55:20	Security	Detailed ...	593	Barrie	PC1
Success Audit	11/10/2007	11:55:11	Security	Detailed ...	592	Barrie	PC1
Success Audit	11/10/2007	11:50:13	Security	Detailed ...	592	Barrie	PC1
Success Audit	11/10/2007	11:49:41	Security	Detailed ...	593	Barrie	PC1
Success Audit	11/10/2007	11:49:39	Security	Detailed ...	592	Barrie	PC1
Success Audit	11/10/2007	11:49:17	Security	Detailed ...	593	SYSTEM	PC1
Success Audit	11/10/2007	11:48:16	Security	Detailed ...	593	NETWORK SER...	PC1
Success Audit	11/10/2007	11:48:13	Security	Detailed ...	592	Barrie	PC1
Success Audit	11/10/2007	11:47:28	Security	Detailed ...	593	Barrie	PC1
Success Audit	11/10/2007	11:47:25	Security	Detailed ...	593	Barrie	PC1
Success Audit	11/10/2007	11:46:46	Security	Privilege ...	576	NETWORK SER...	PC1
Success Audit	11/10/2007	11:46:46	Security	Logon/Lo...	528	NETWORK SER...	PC1

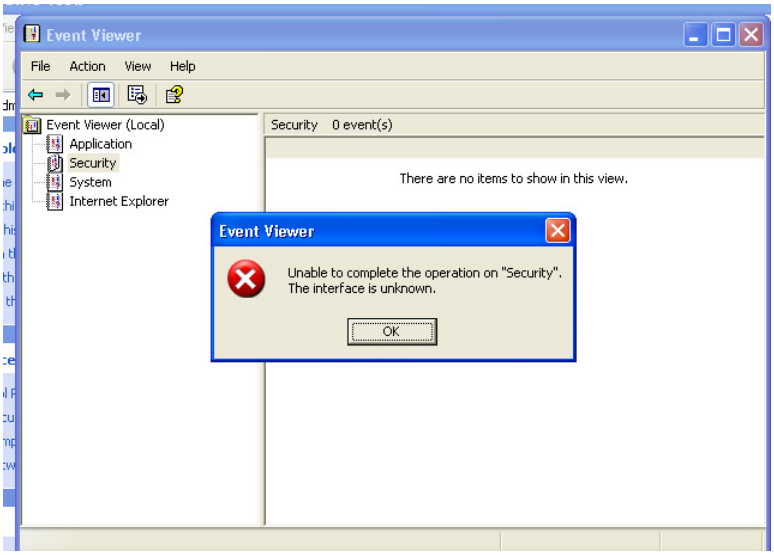
Now the computer is restarted.

It was noted that there was a noticeable delay when the system booted back up, this happened just before the login screen appeared.

Another check of the Services reveals that the Event Service has been stopped.

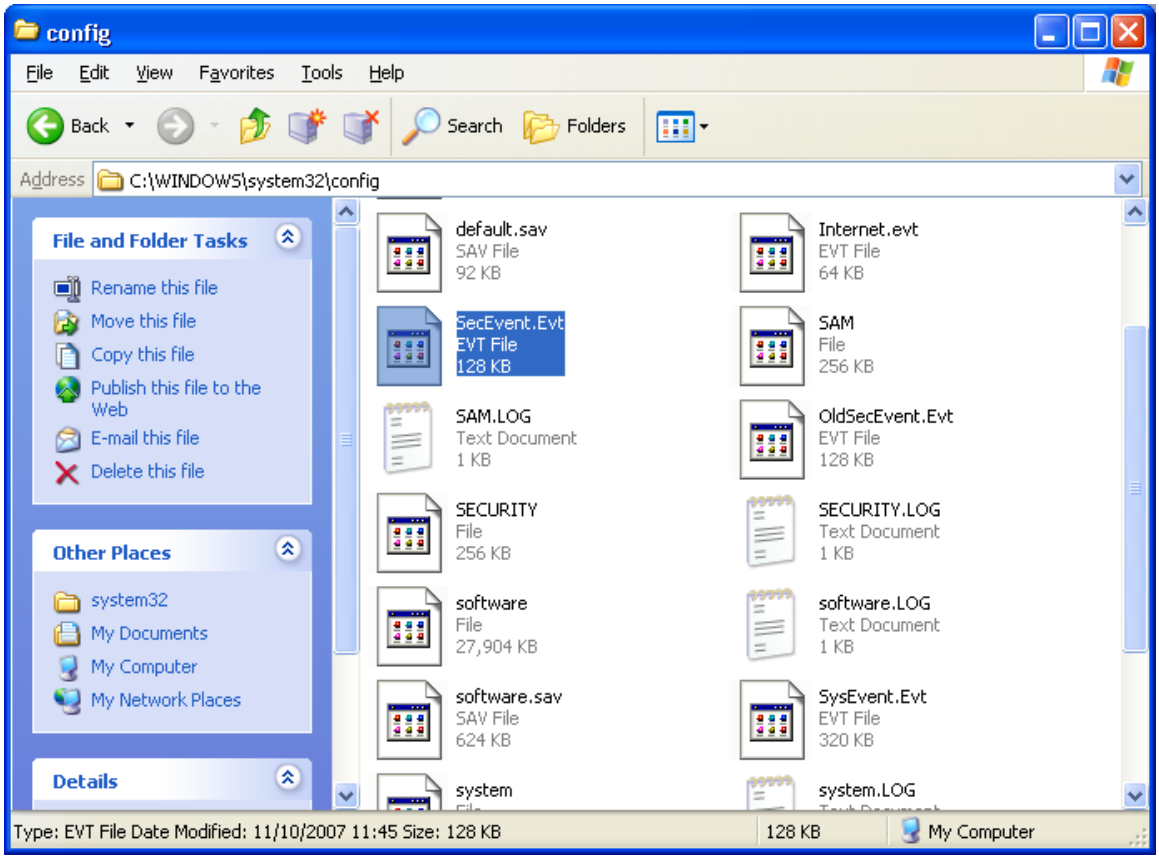


And the Event Viewer produces the following error when trying to access it:



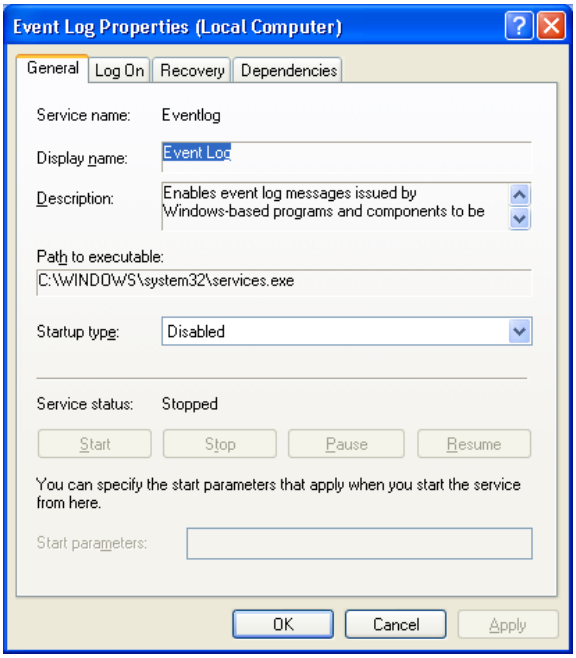
We now have full control over the original ‘SecEvent.Evt’ file and as such it is replaced with the copy that was previously made.

The original file is kept and renamed to ‘OldSecEvent.Evt’, while ‘Copy of SecEvent.Evt’ is renamed to replace its original.

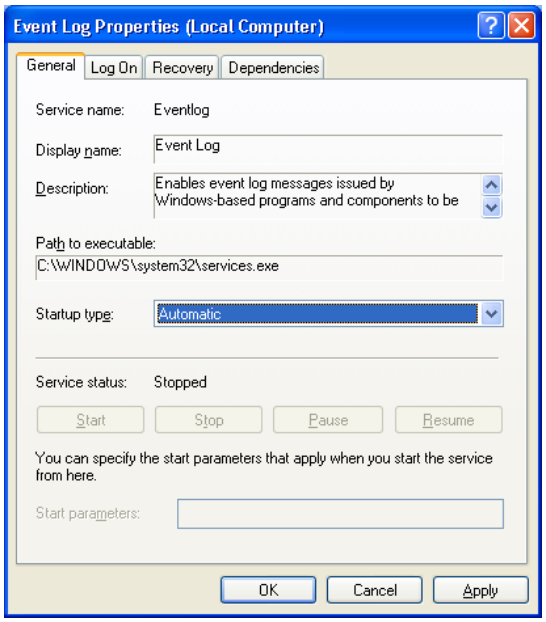


The Event Service is then restarted (this can be done with the need to reboot the machine).

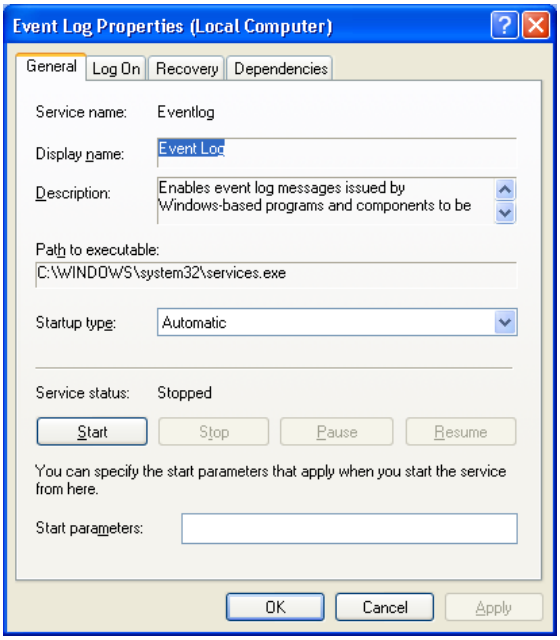
First the 'Startup type' must be changed from 'Disabled'



It will be set to ‘Automatic’; this will enable it to function as normal after the experiment is complete.

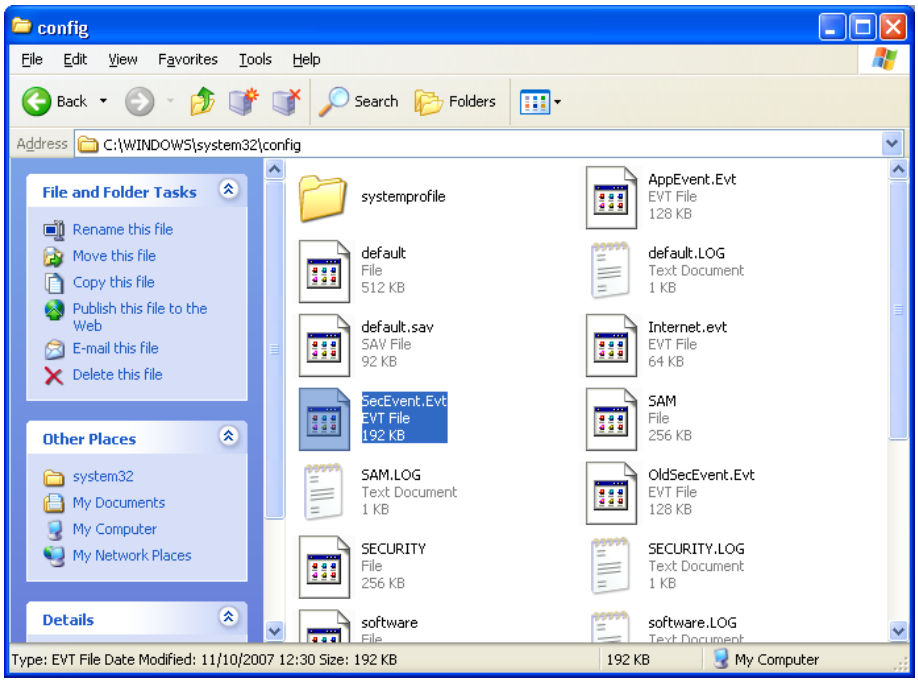


After the changes have been ‘Applied’ it enables the ‘Start’ button.

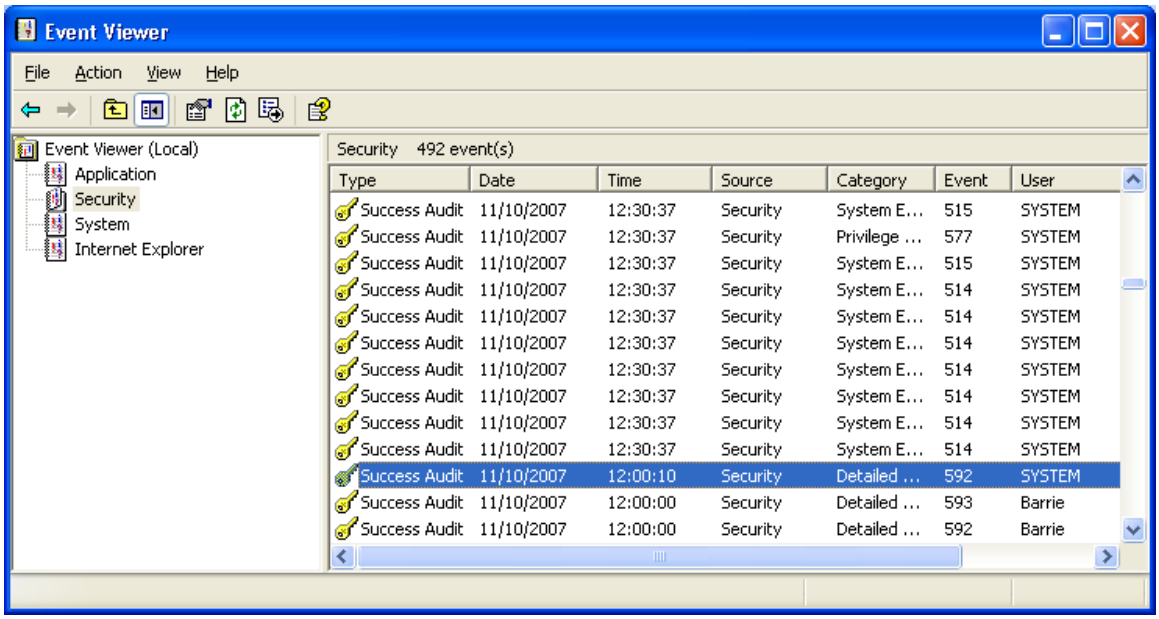


Clciking the start button will restart the service.

Another look at the ‘c:\windows\system32\config’ folder shows that quite a bit of information has been written to the log file, it has increased in size by 74Kb.



And another look at the Security Log reveals that it now contains 492 events.



It is also noted that it has continued to write to the log file as if nothing has happened.

There will probably be an event that signifies that the Event Service has been stopped and started. This will require further investigation.

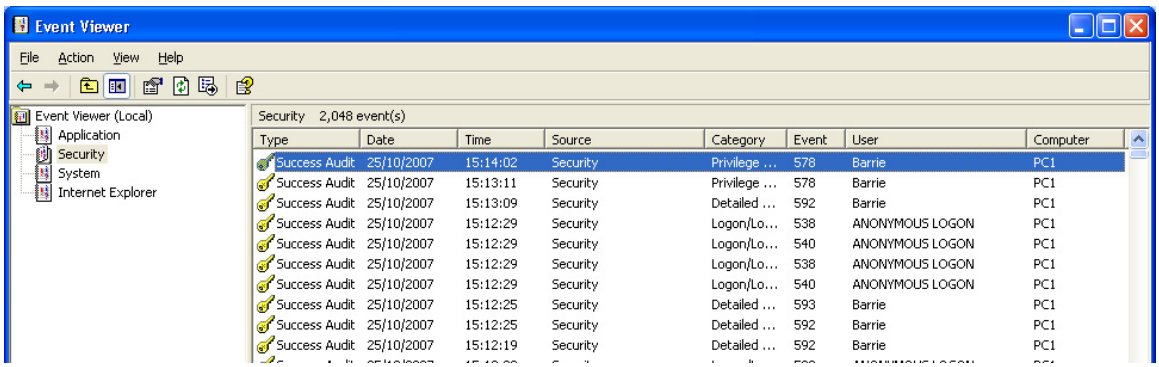
Swap the security log from one computer to another

Computer Name	Local Users
PC1	Barrie
	Donald
PC2	Barrie

Task:

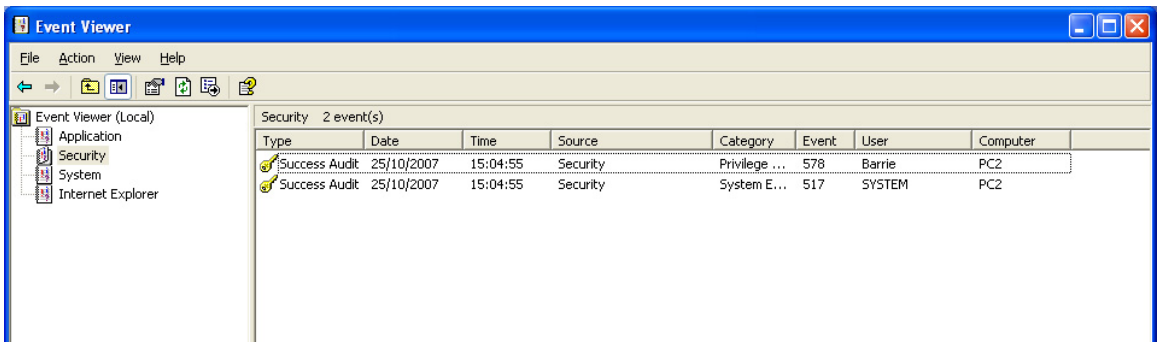
To inject the security log from PC1 into PC2 and note any effects or problems

Screen shot of security log on PC1



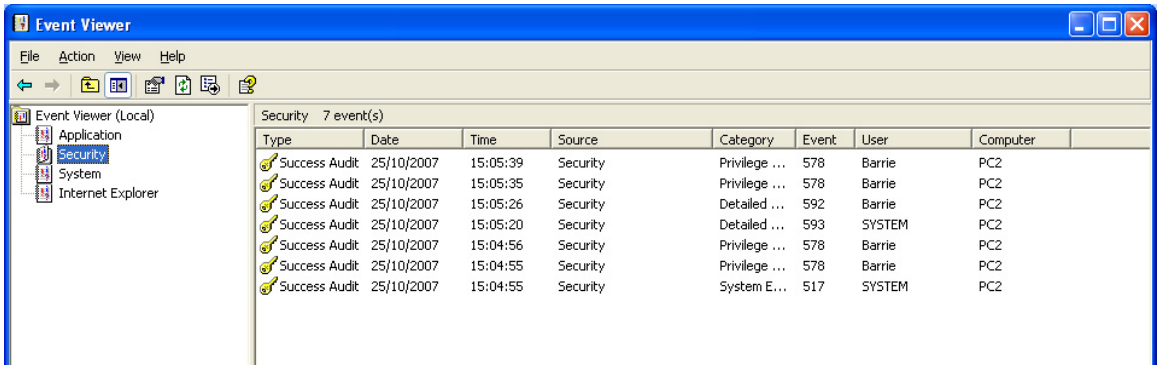
Then started working on PC2

Cleared the Security log on PC 2

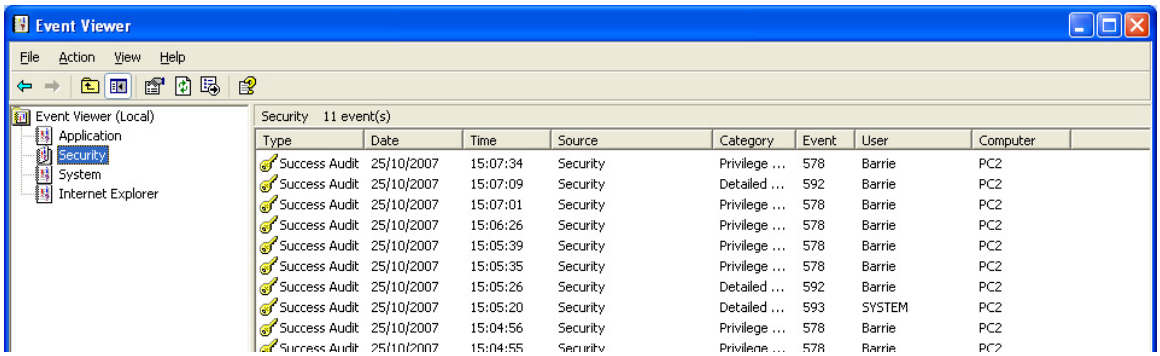


Privileged object operation:
Object Server: EventLog
Object Handle: 11670928
Process ID: 748
Primary User Name: PC2$
Primary Domain: WORKGROUP
Primary Logon ID: (0x0,0x3E7)
Client User Name: Barrie
Client Domain: PC2
Client Logon ID: (0x0,0x109A9)
Privileges: SeSecurityPrivilege

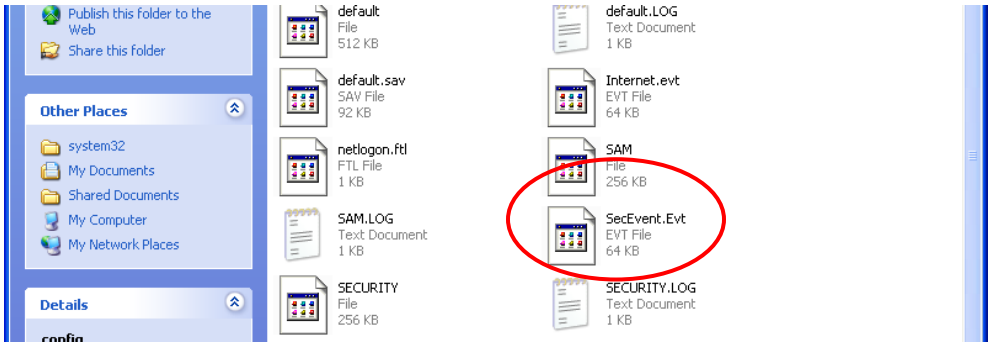
Started up Word



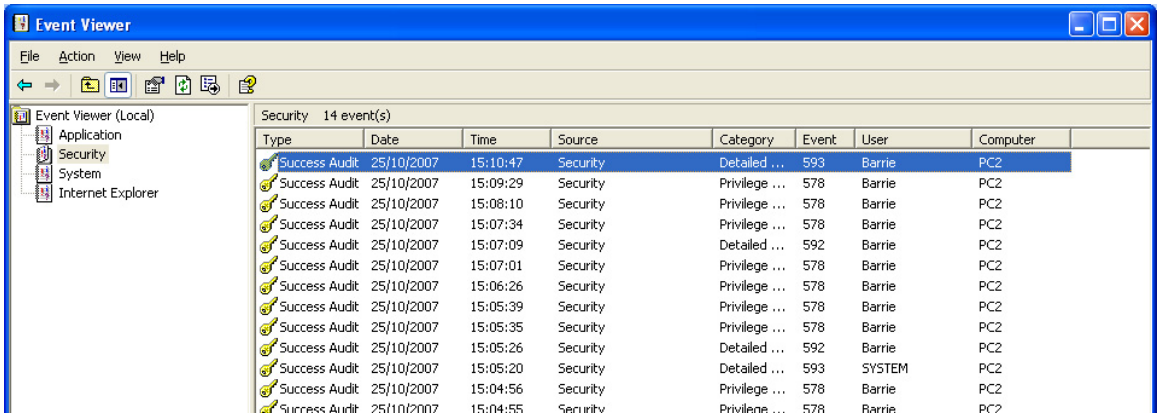
Navigated to c:\windows\system32\config

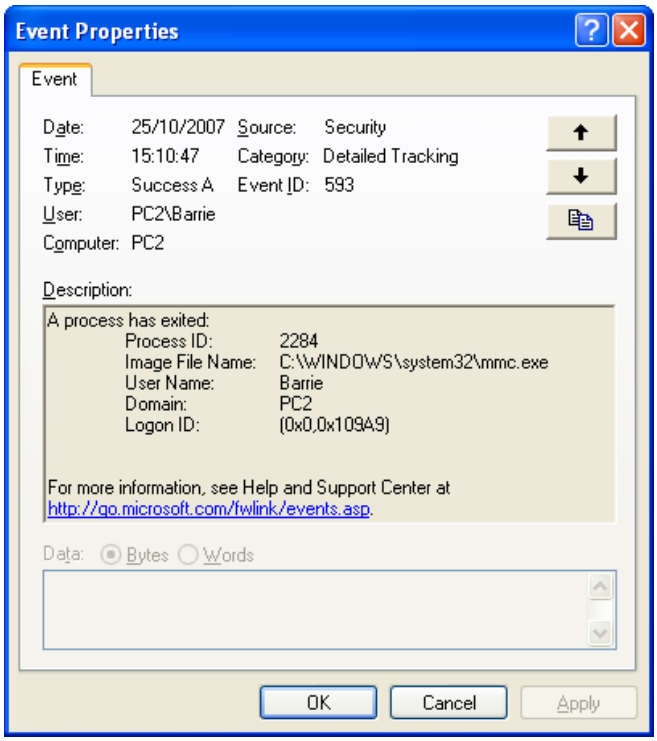


Took screen grab of folder



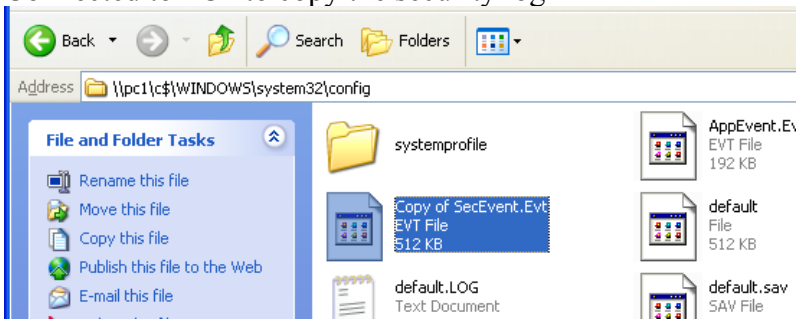
Turned off the event log



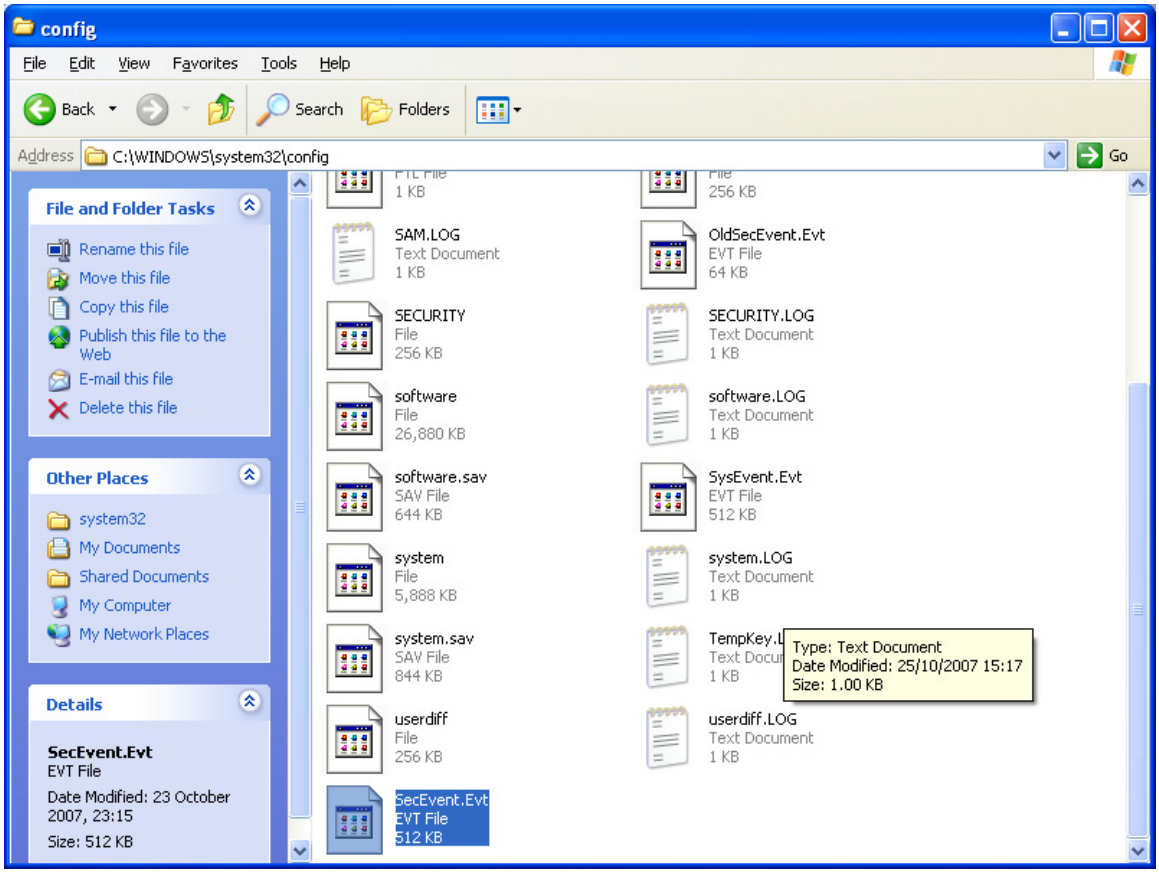


Restarted the computer.

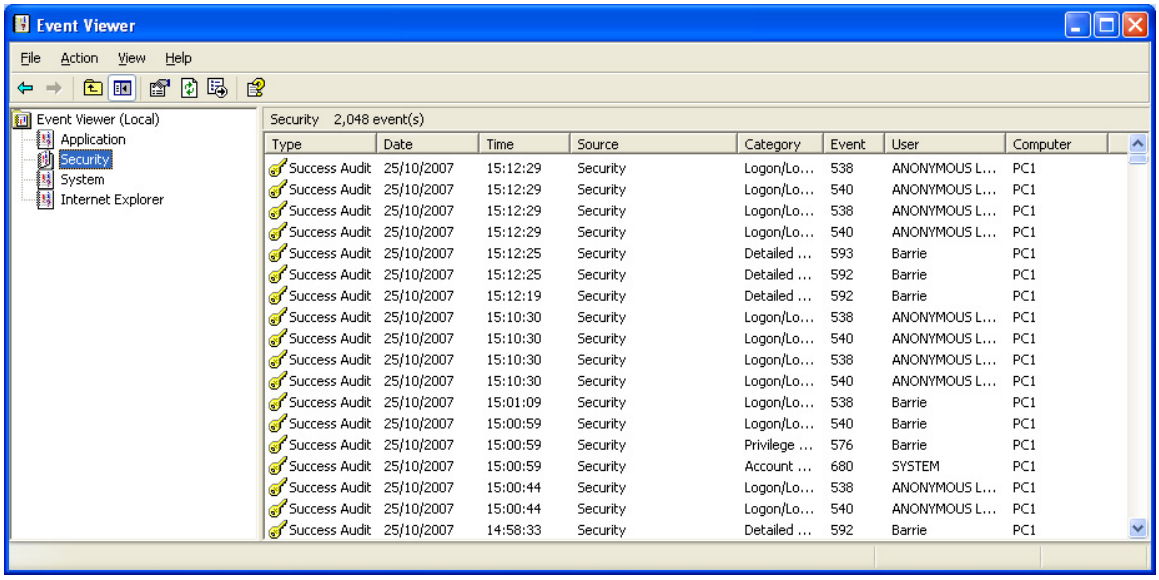
Connected to PC1 to copy the security log



Copied it across from PC1 to PC2



And then restarted the event service



Initial Diagnosis:

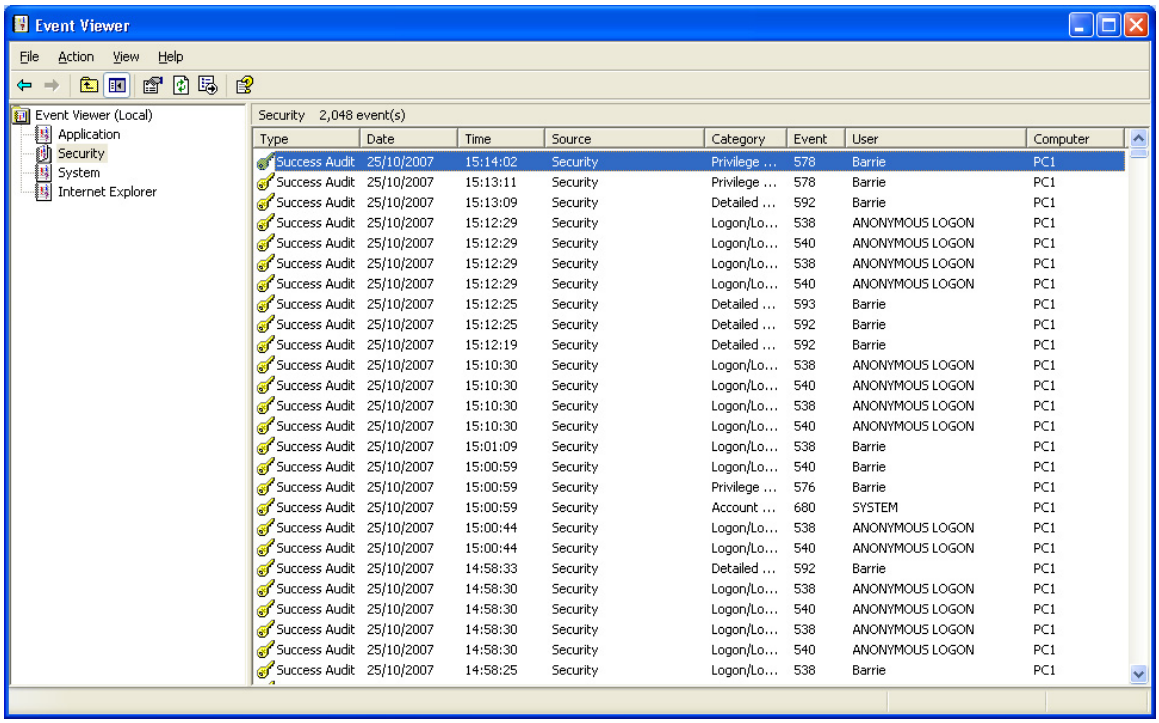
Event log started with no problems or errors

One point to note is that the PC2 usernames are now identifiable and the PC1 usernames have been changed to ANONYMOUS LOGON. Also, the log file has maintained the computer name of PC1 in the column on the right hand side.

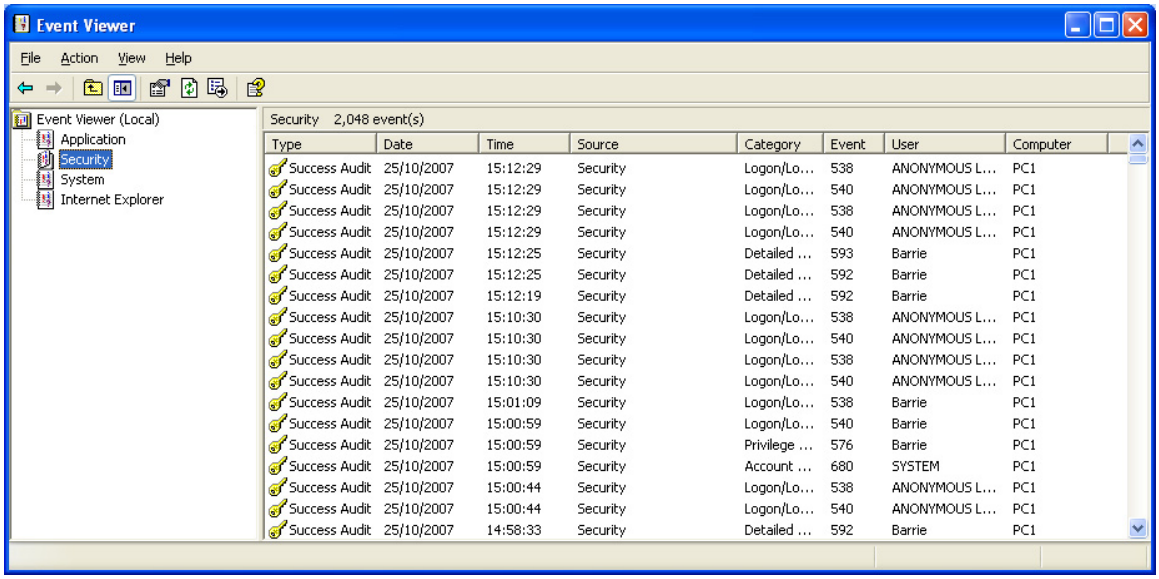
PC2 was then rebooted and its security log examined again… The log was still exactly the same as before, this has presented a problem for further analysis, as no new events have been logged.

Screen shots of the same log on 2 different computers

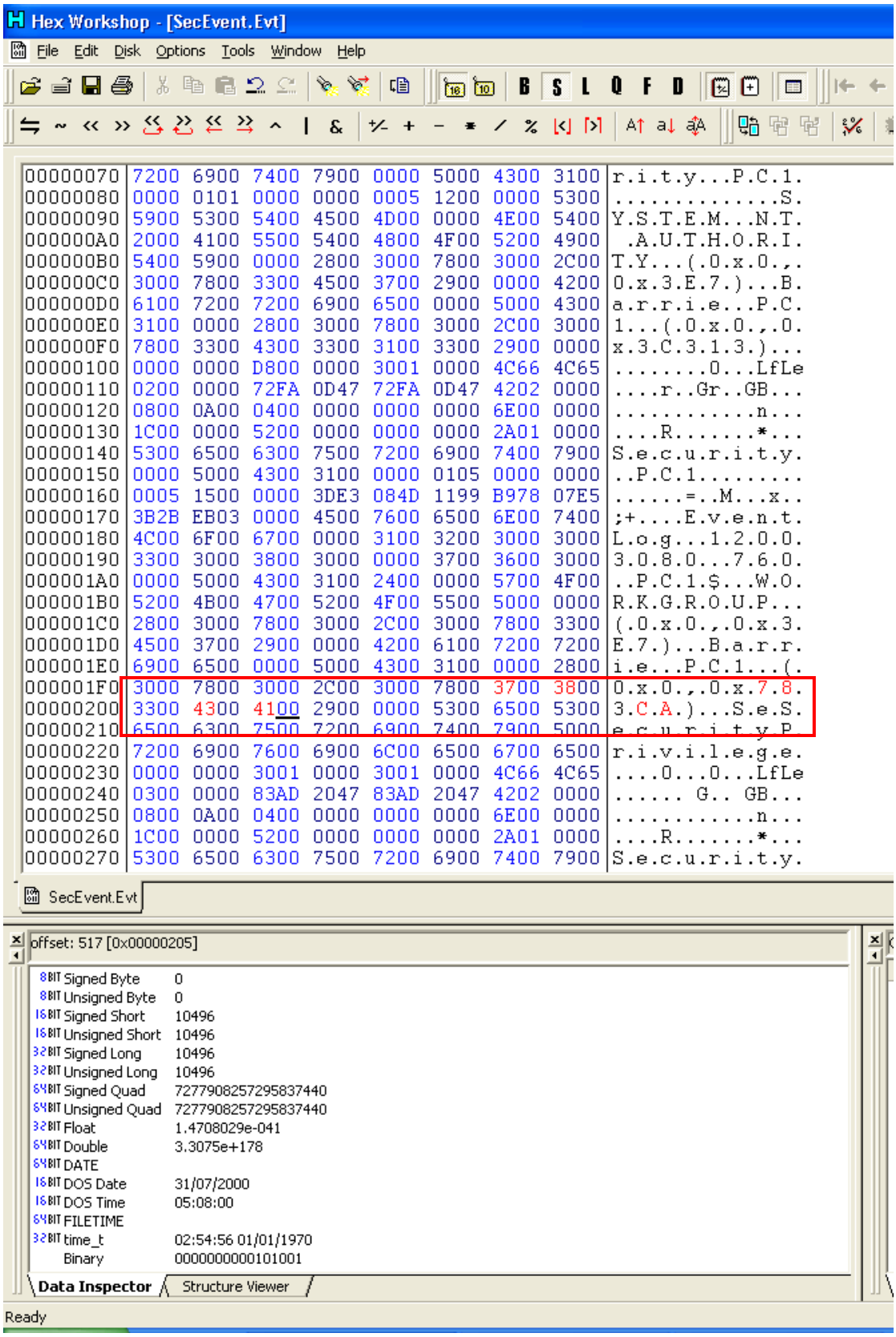
PC1

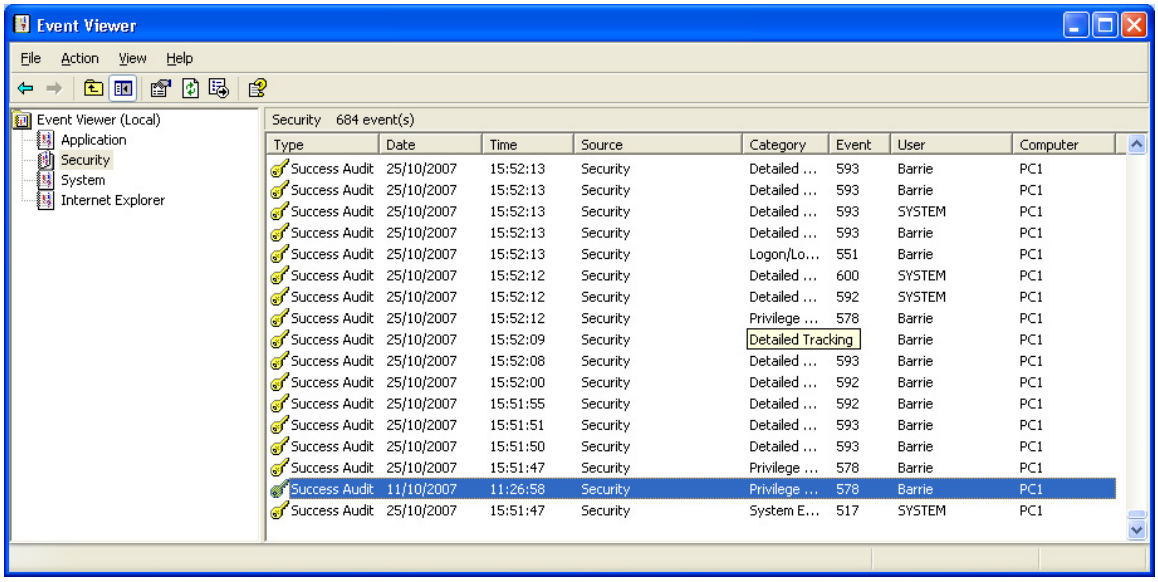


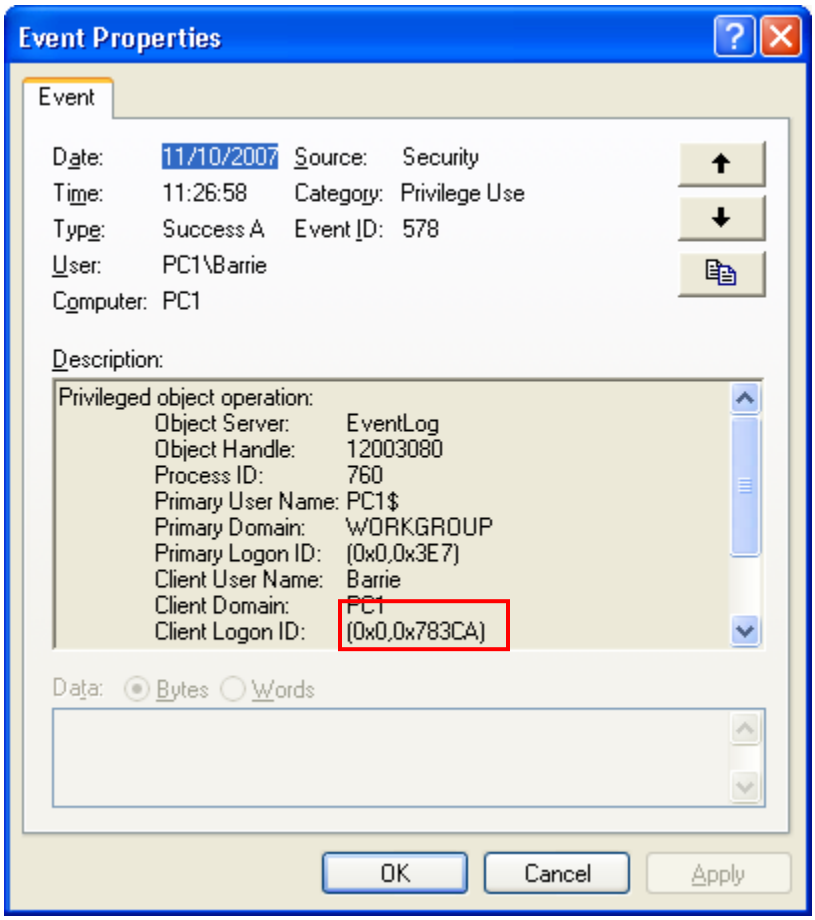
PC2



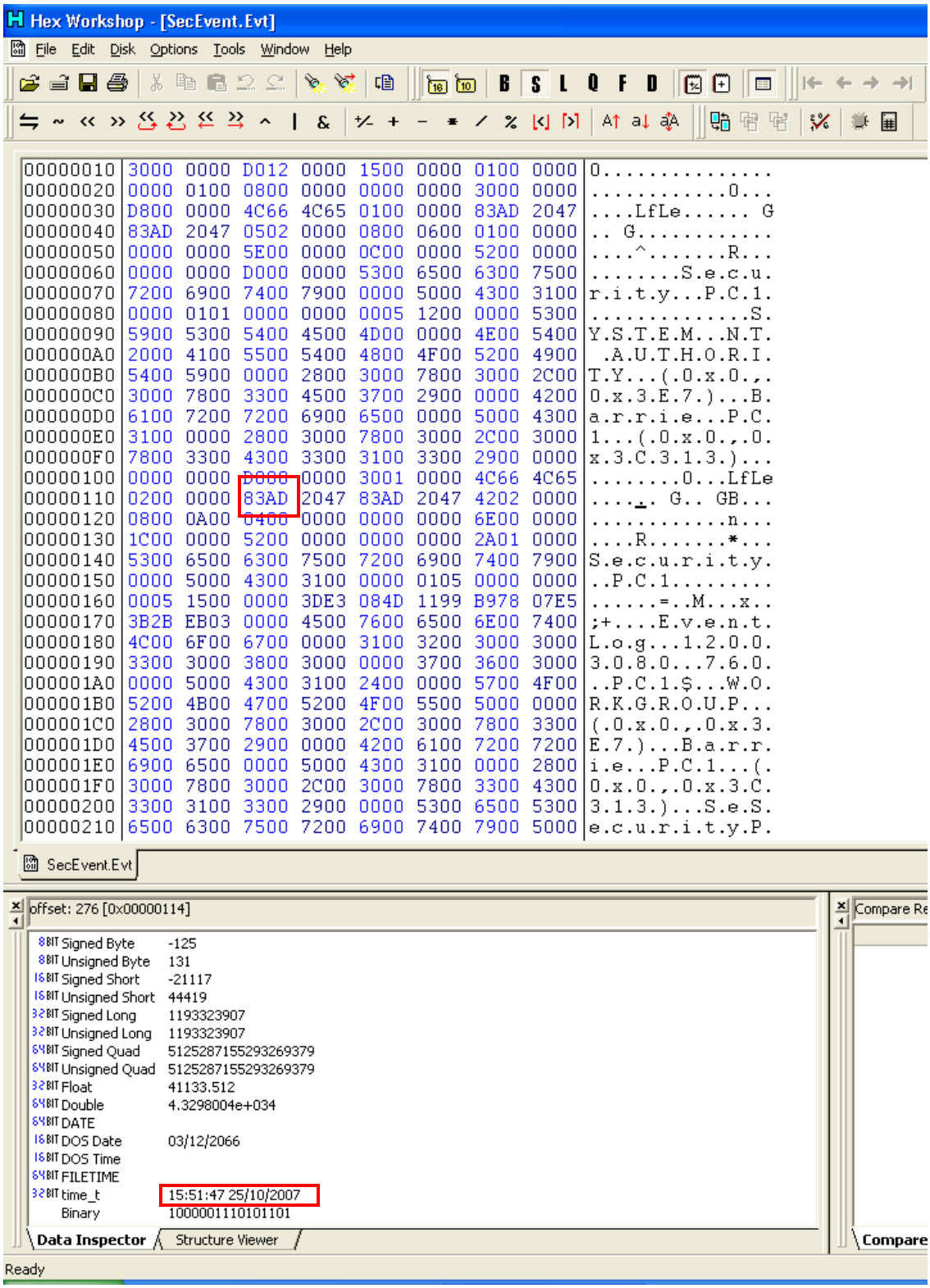
Modify the ‘logon ID’ to try and change the computer name and user name

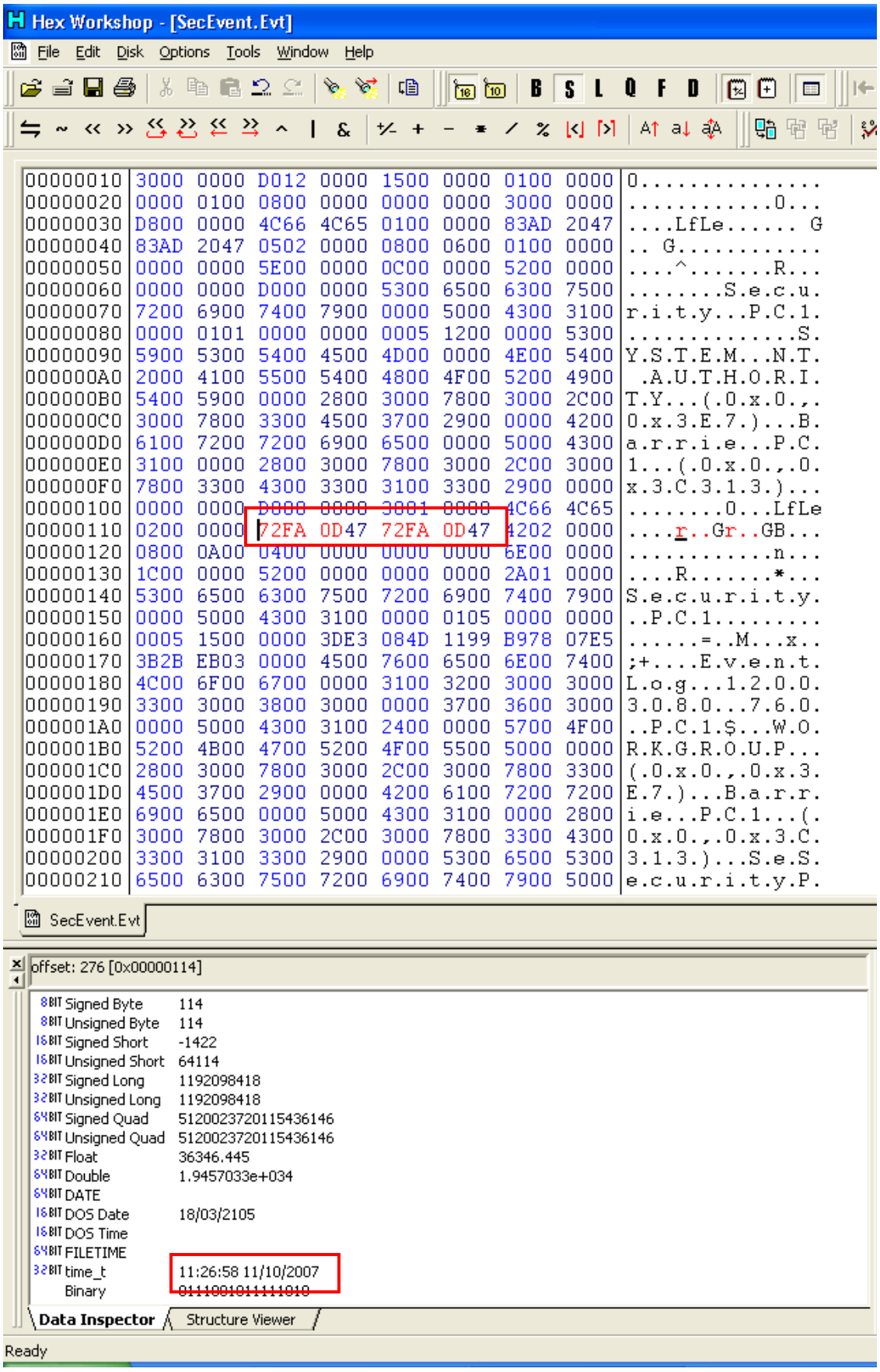


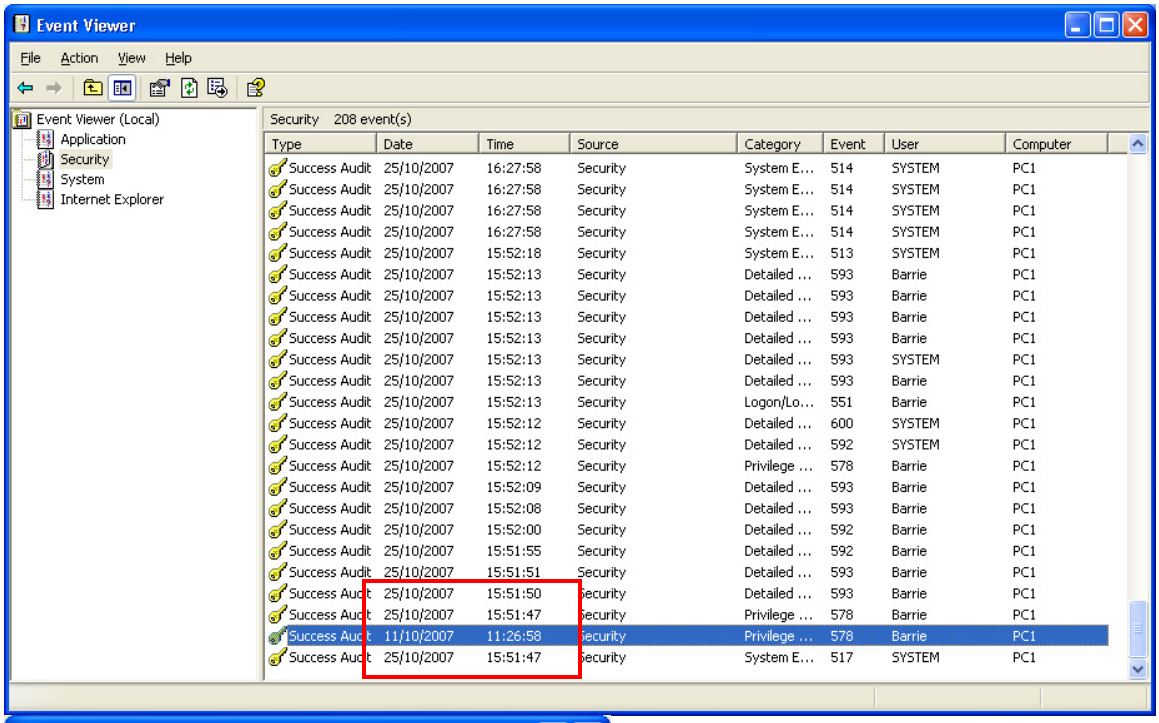


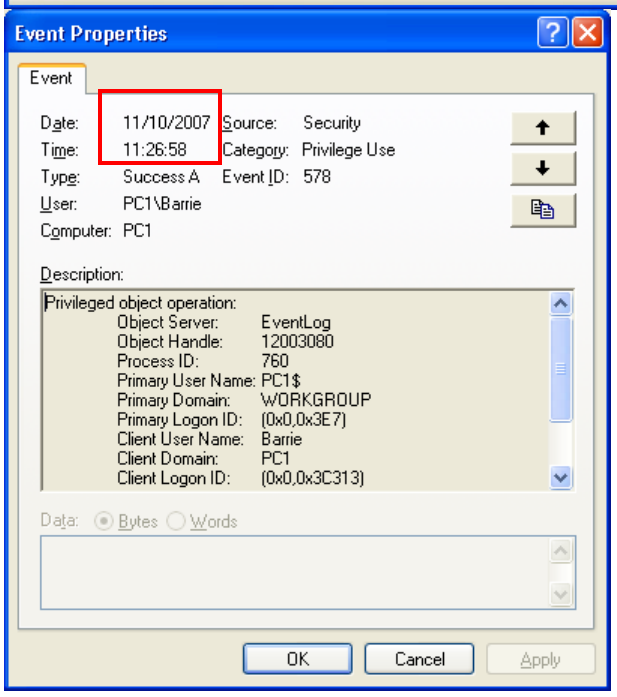


Modify the 32 bit time

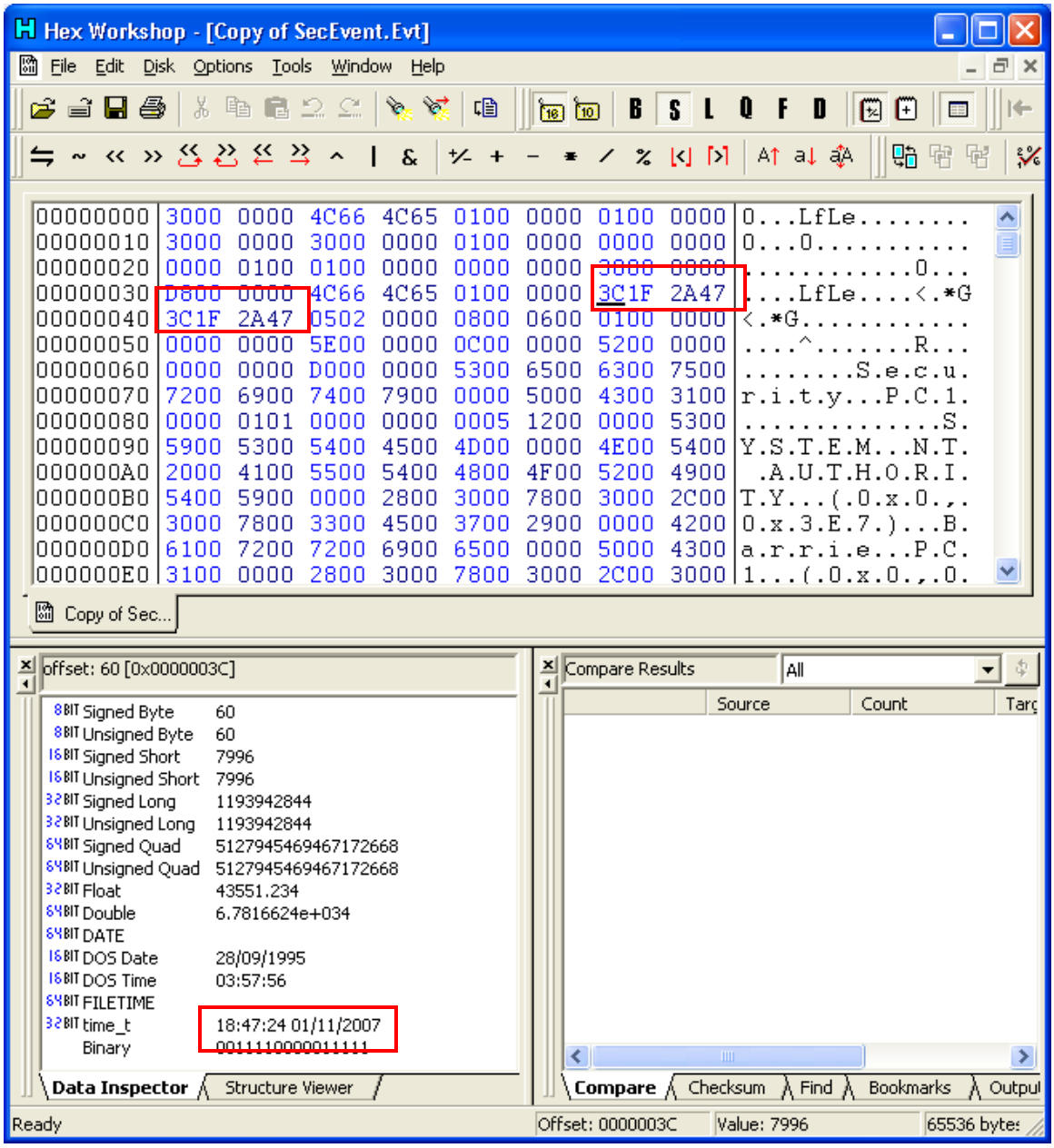








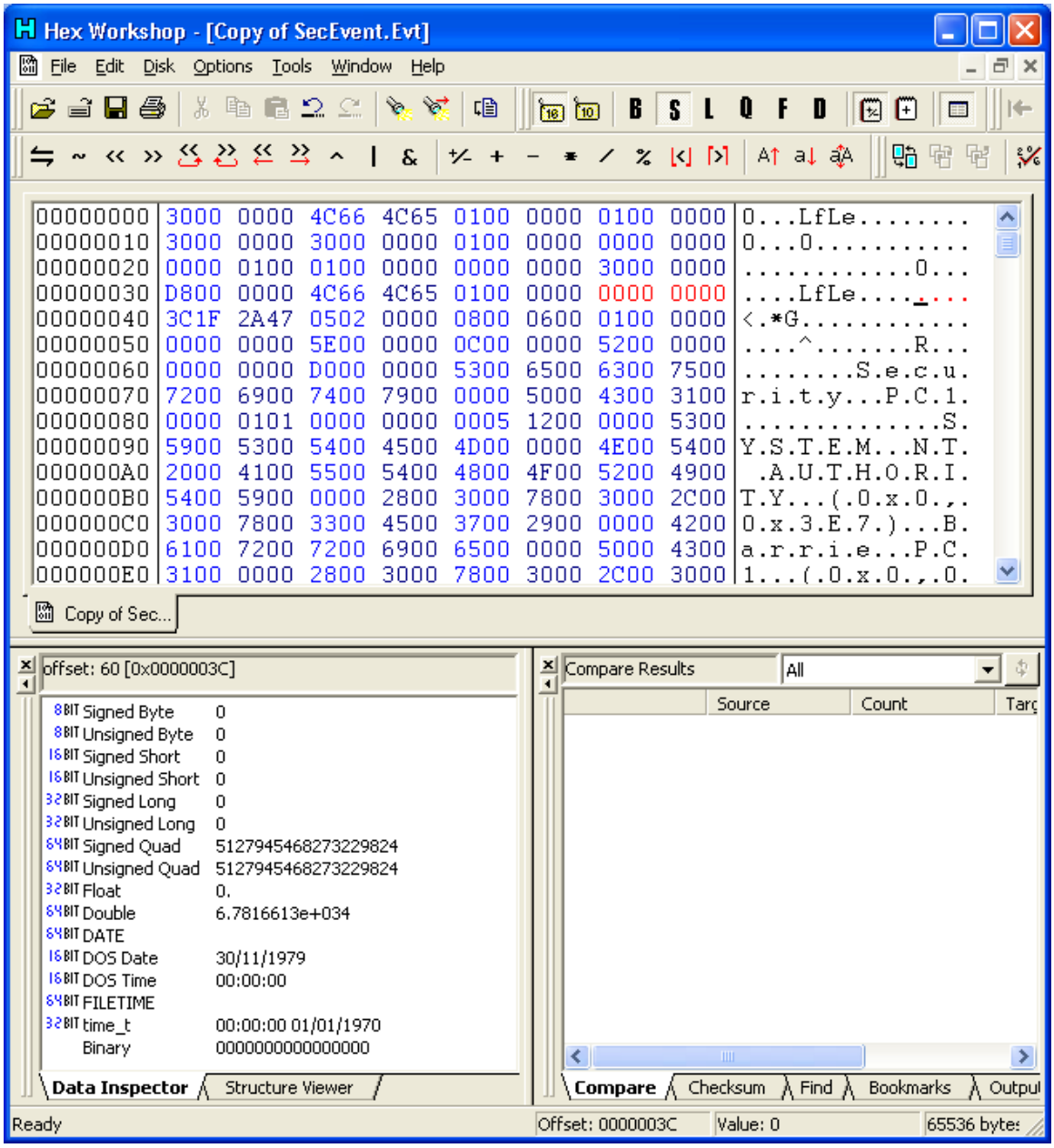
How the event log stores the time

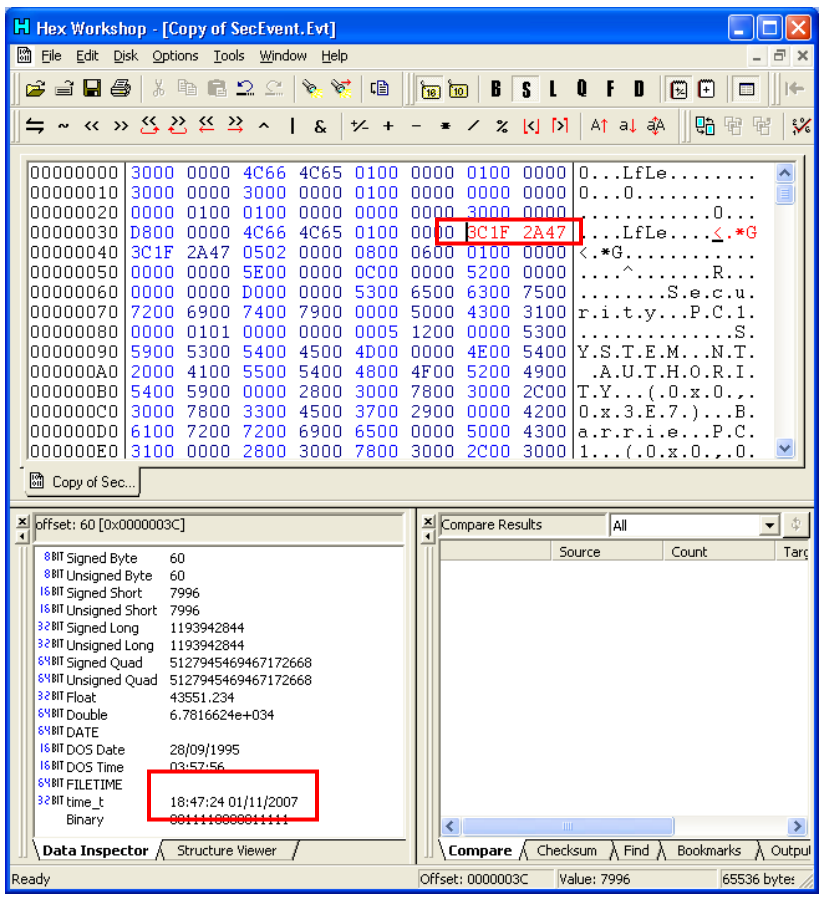


As previously discovered the event time is contained within the above noted 32bits immediately followed by a repetition of the value.

By resetting the hex value to 0000 0000 it was noted that the time and date counted up from 00:00:00 01/01/1970.

Note: Research has shown that storing a date in format will prove to be problematic in the year 2038. People are referring to this as the Y2K38 Bug.





The screenshot on the left shows the original hex value (3C 1F 2A 47), the screenshot on the right shows the first set of hex values has been increased by 1 (3D 1F 2A 47). It is noted that the 32 bit time has increased by 1 second.

Further investigation is required to calculate a hex value from a given time & date.

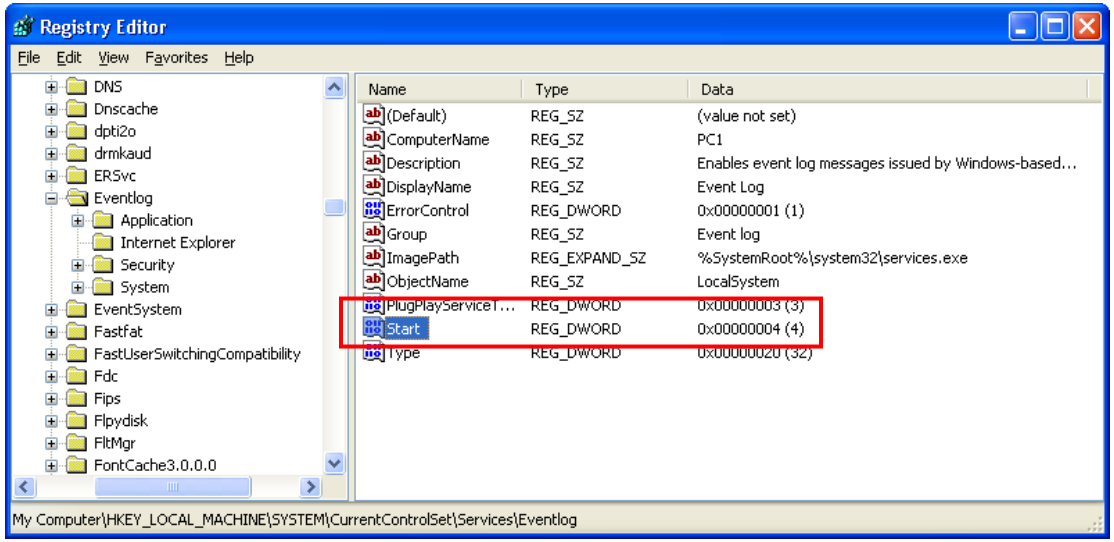
Automatically replace the ‘SecEvent.Evt‘ file

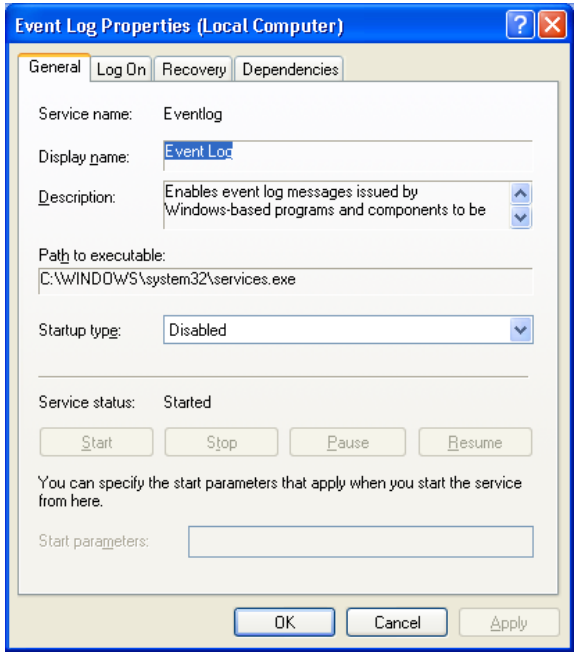
Displaying the registry value

The following code will display the current start-up settings for the event log.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Win32;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            RegistryKey key =
Registry.LocalMachine.OpenSubKey("System\\CurrentControlSet\\Services\\EventLog");
            if (key.GetValue("Start") != null)
            {
                // The value exists;
                Console.WriteLine((int)key.GetValue("Start"));
            }
            Console.ReadLine();
        }
    }
}
```

The registry value has the following 3 meanings:
2 = Automatic
3 = Manual
4 = Disabled





Modifying the registry

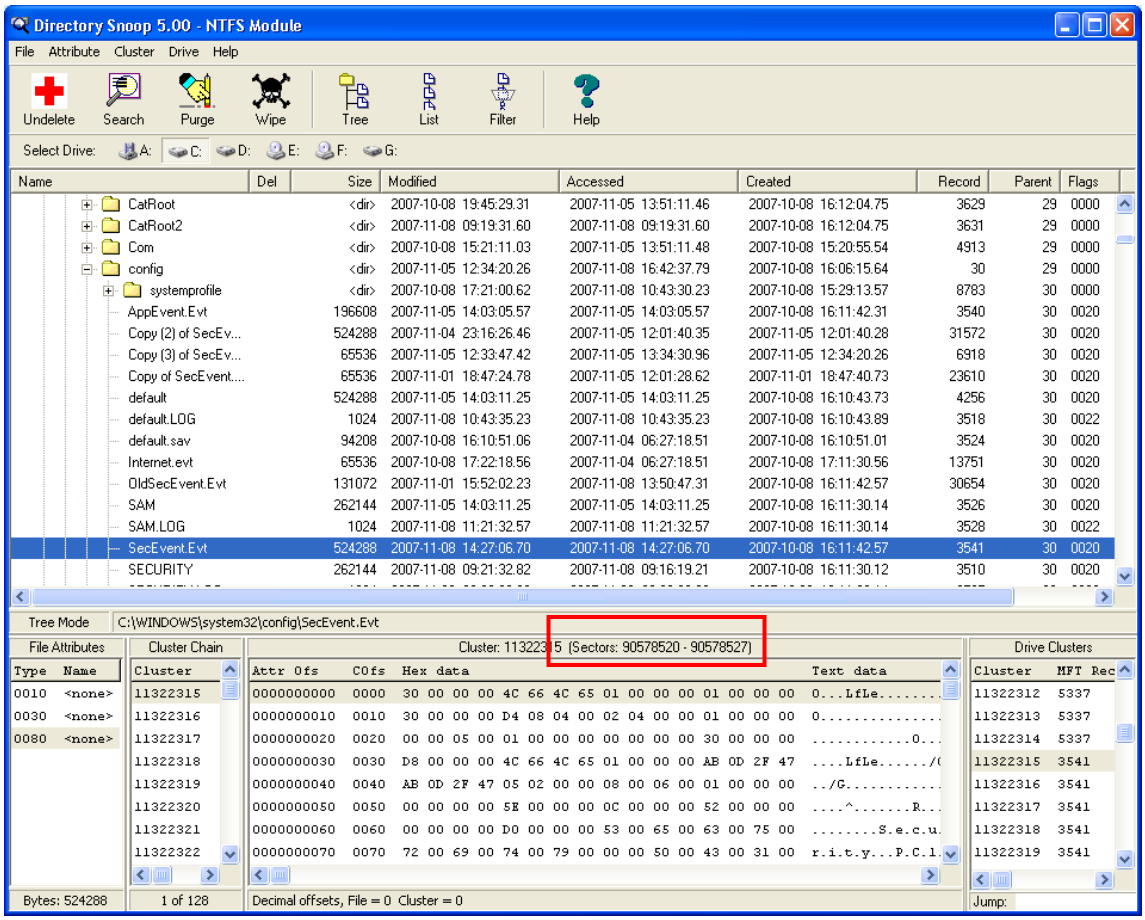
The following code will display the current start-up settings for the event log and allow the user to modify it.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Win32;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            RegistryKey key =
Registry.LocalMachine.OpenSubKey("System\\CurrentControlSet\\Services\\EventLog", true);
            if (key.GetValue("Start") != null) //Check the value actually exists
            {
                // The value exists
                Console.WriteLine("Its value started at: " + (int)key.GetValue("Start"));
                Console.WriteLine("Please select a new value (2, 3 or 4): ");
                String newNumString = Console.ReadLine();
                int newNum = Convert.ToInt32(newNumString);
                if (newNum >1 && newNum <5)
                {
                    key.SetValue("Start", newNum);
                }
                Console.WriteLine("Its new value is: " + (int)key.GetValue("Start"));
            }
            Console.ReadLine();
        }
    }
}
```

Automatically replacing ‘SecEvent.Evt’

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Win32;
using System.IO;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            RegistryKey key1 =
Registry.LocalMachine.OpenSubKey("System\\CurrentControlSet\\Services\\EventLog", true);
            RegistryKey key2 =
Registry.LocalMachine.OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce", true);
            int newNum = Convert.ToInt32(key1.GetValue("Start"));
            if (newNum == 2) //Check the value is set to 'automatic'
            {
                if (key2.GetValue("MyApp") == null) // The value does not exists
                {
                    key2.SetValue("MyApp","c:\\MyApp.exe");
                    key1.SetValue("Start",4);
                }
                newNum = 0;
            }
            if (newNum == 4) //Check the value is set to 'disabled'
            {
                File.Replace("c:\\windows\\system32\\config\\dummy.dat",
"c:\\windows\\system32\\config\\SecEvent.Evt", "c:\\windows\\system32\\config\\OldSecEvent.Evt", false);
                key1.SetValue("Start", 2);  // Restart event log service???
            }
        }
    }
}
```

Master File Tables



The above screenshot shows the sectors of the SecEvent.Evt file

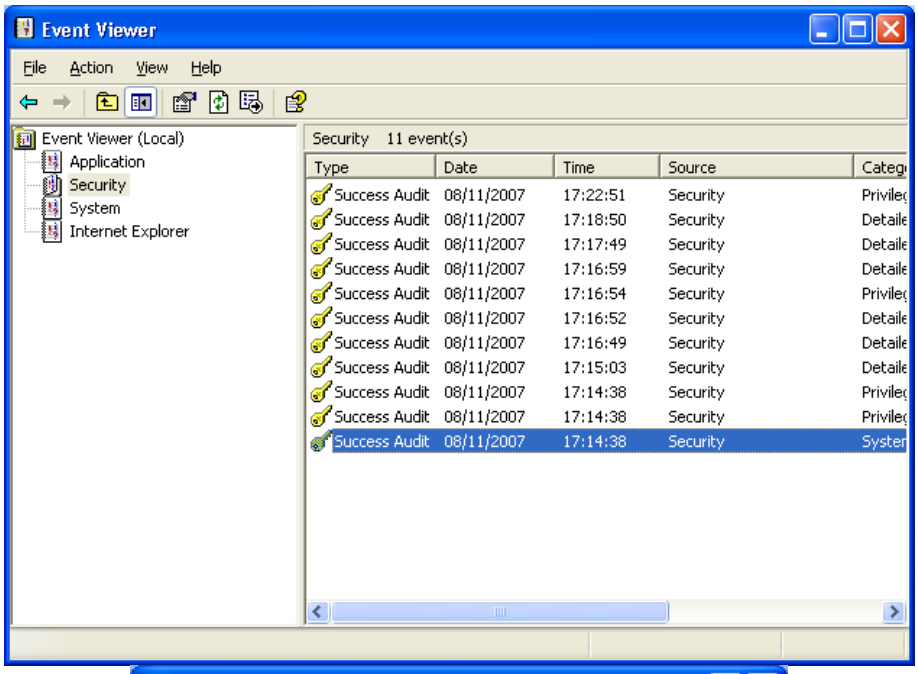
Modify the sectors

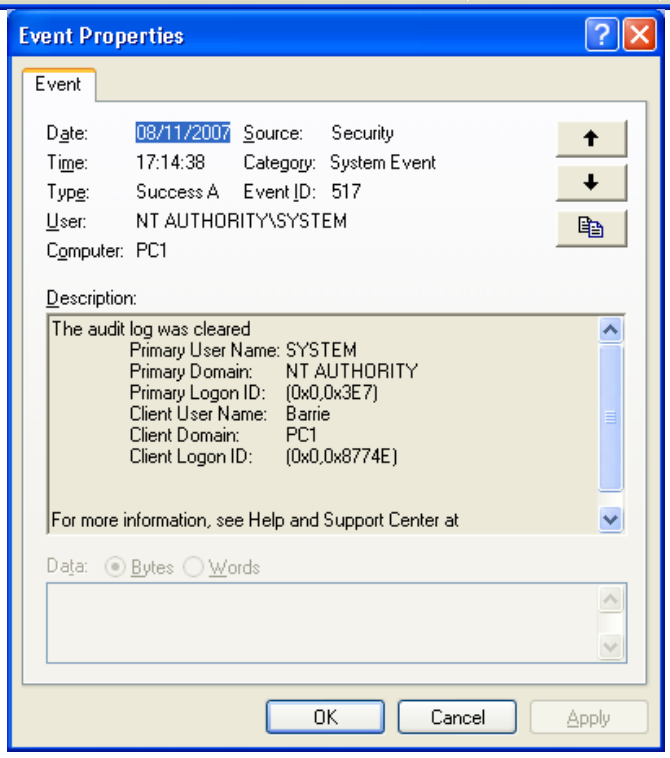
- Produce a pice of code that directly modifys the Hex Data contained with the drive secotrs.
- This will bypass the file locking.
- This process would be slower, as it would need to copy the entire log file, but would be harder to detect.

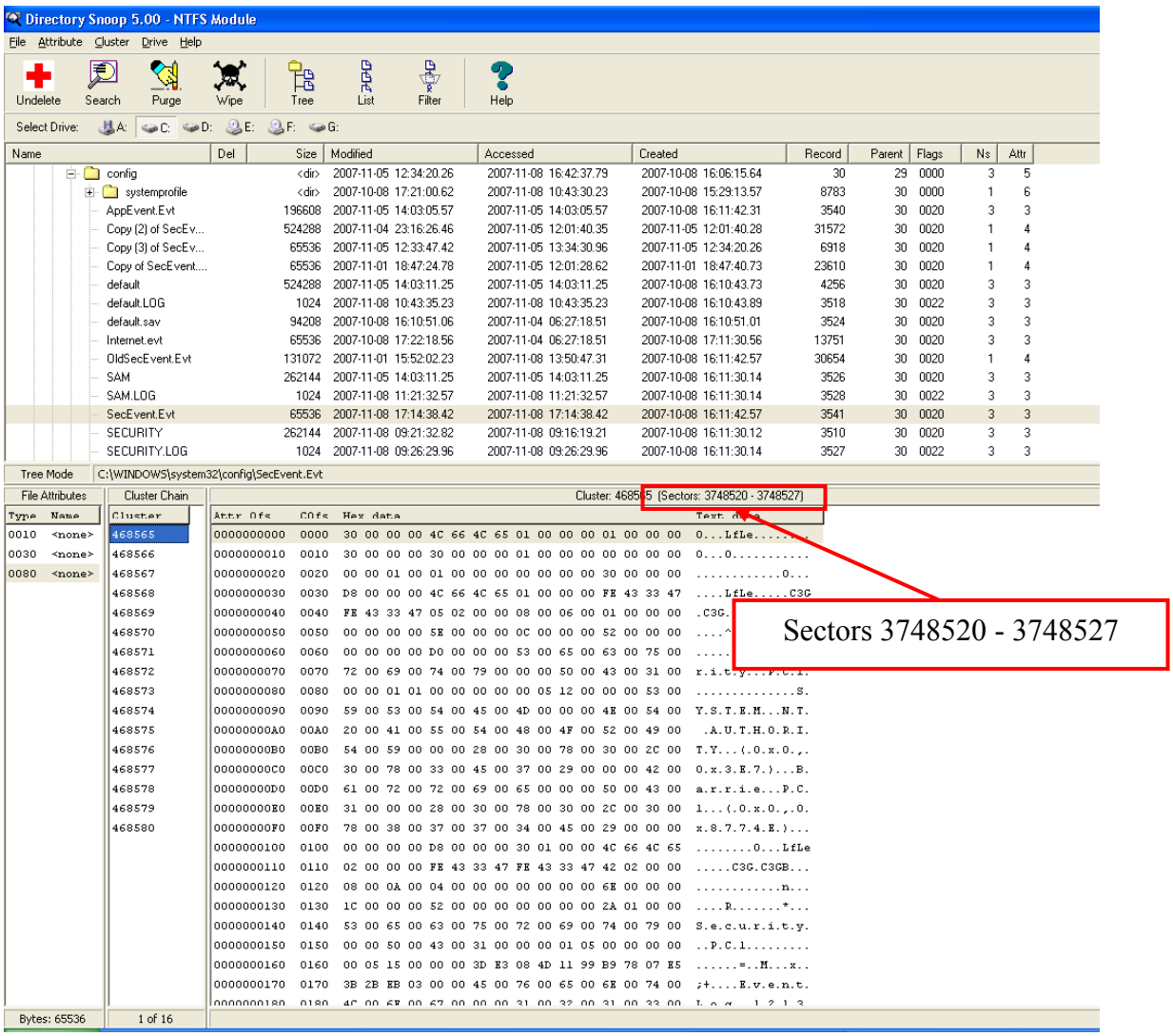
Swap the pointers

- Produce a piece of software that modifys the pointers to the cluster chains
- This would bypass the file locking.
- This process would be fast since it only has to change a coulpe of dozen Hex values, however, it would be easier to detect since the ‘new log’ cluster chains would start nearer to the end of the disc.

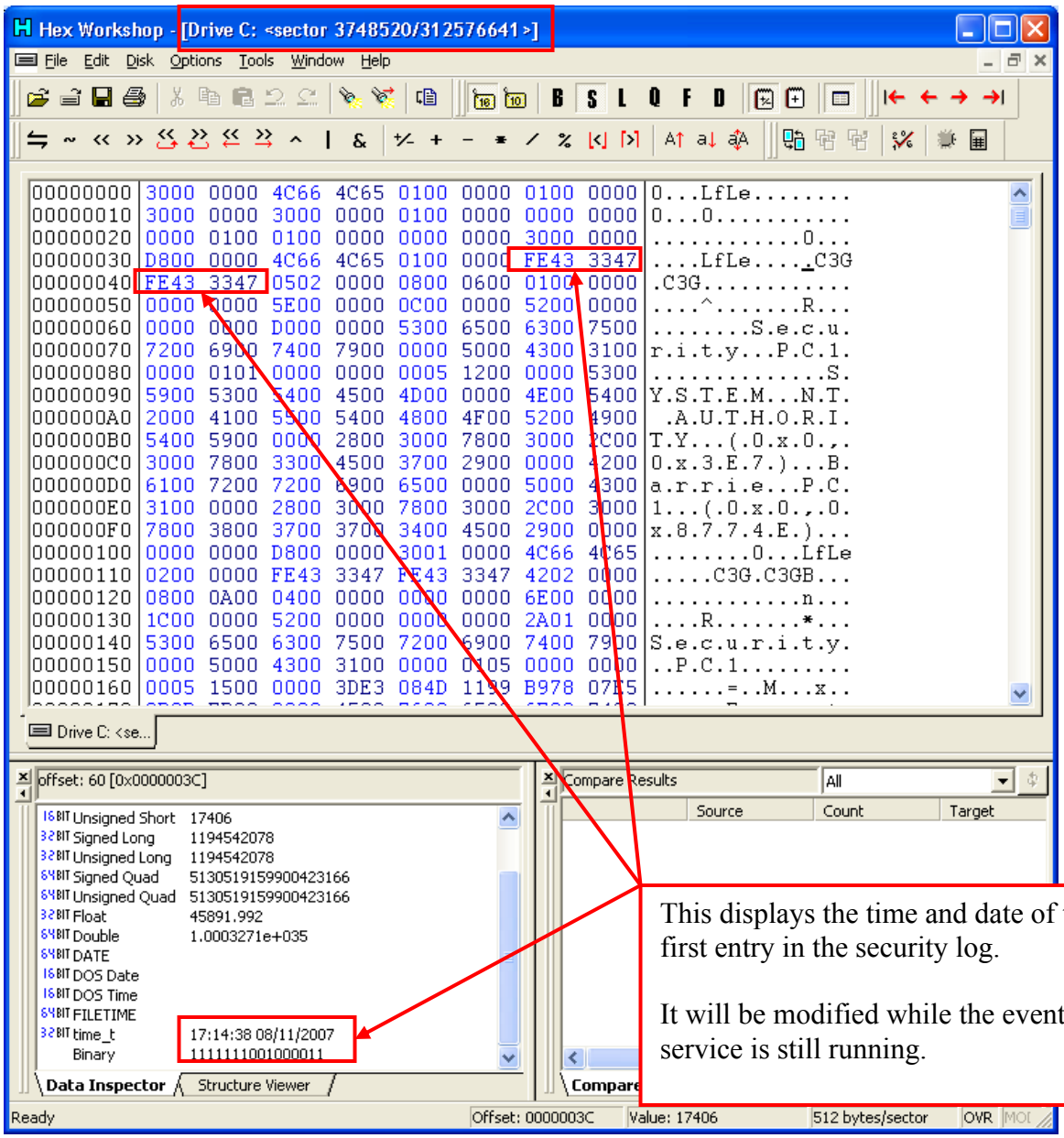
Modifying the Sectors





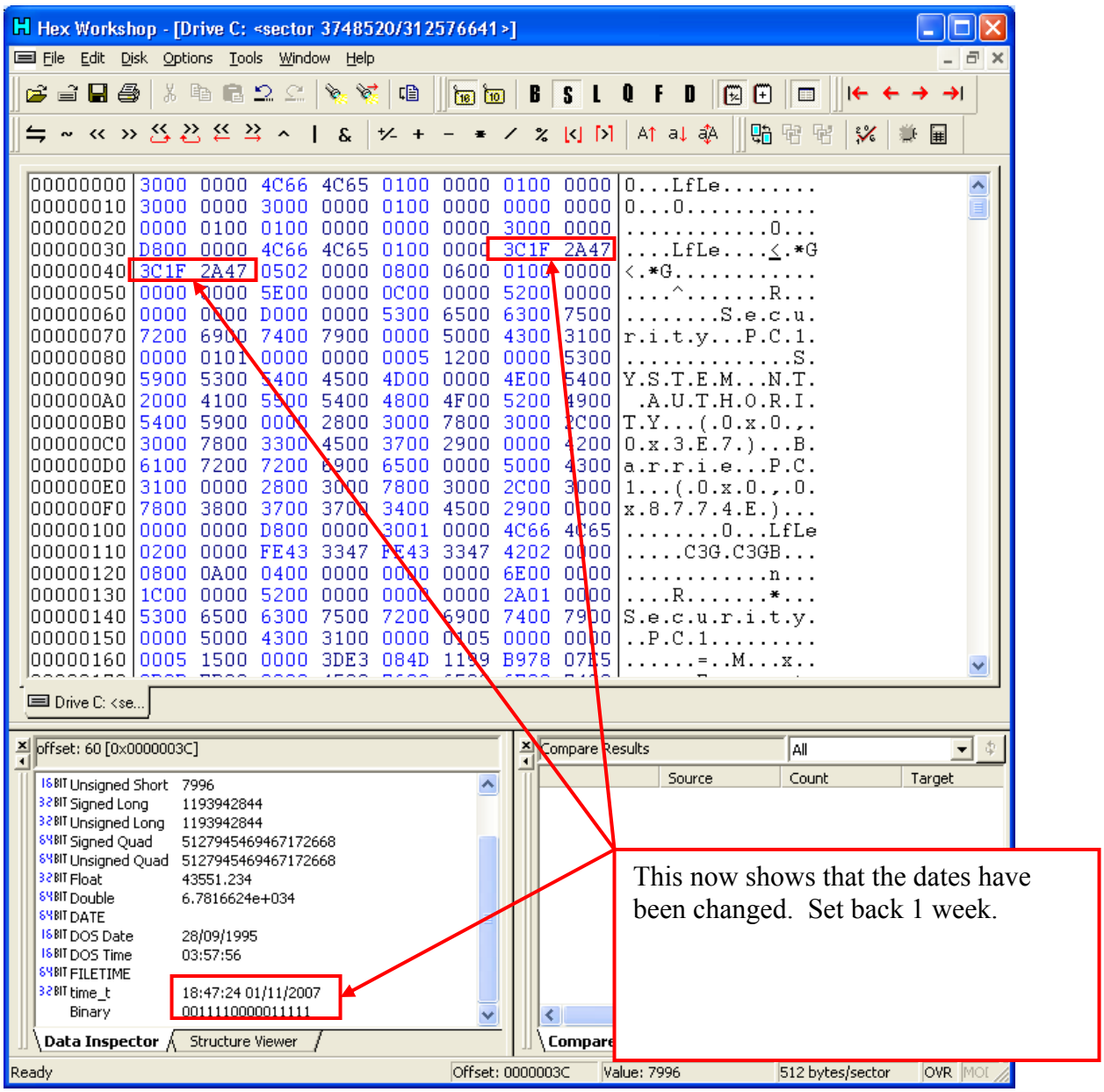


The starting sector 3748520 is then put into hex editor.



This displays the time and date of the first entry in the security log.

It will be modified while the event service is still running.



These modifiactions were then saved to disc and directory snoop was refreshed, this then showed that the data on the drive had been modified, however when the event viewer was opened up it did not show any changes. This must mean that it does not read from the file after it has loaded up.

The system was then restarted.

Directory Snoop was opened up and pointed towards the SecEvent.Evt file, this showed that the time and date had been set back to their original value. This was varified with opening up the event viewer.

Perhaps it only stores 'new events' in memory. That is events that have happened after the system was started.

So once again the Hex editor was used to modify the time and date of 'old events' and this was verified as being changed on the hard drive using directory snoop.

The system was then restarted.

A quick look with the event viewer shows that the times of the events have been set back to their original value.