CYBER-FORENSIC LOG ANALYSIS

Assaad Sakha

A thesis

in

Concordia Institute for Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science (Information System Security)
Concordia University
Montréal, Québec, Canada

September 2008

Your file Votre référence
ISBN: 978-0-494-45501-2
Our file Notre référence
ISBN: 978-0-494-45501-2

Abstract

Cyber-Forensic Log Analysis

Assaad Sakha

Forensic examination of logs plays a big role in modern computer security. Due to the sheer amount of data involved and the evolving complexity of computer systems, the forensic examination of logs is a time consuming and daunting task. Information stored in logs of a computer system is of crucial importance to gather forensic evidence of investigated actions or attacks against the system. Analysis of this information should be rigorous and credible, hence it lends itself to formal methods. In this thesis, we propose a model checking approach to the formalization of the forensic log analysis. In order to provide a structure to the log events, we express each event as a term of a term algebra. The signature of the algebra is carefully chosen to include all relevant information necessary to conduct the analysis. Properties of the model are expressed as formulas of a logic having dynamic, linear, temporal, and modal characteristics. A tableau-based proof system is provided for this logic upon which a model checking algorithm can be developed. In order to illustrate the proposed approach, the Windows XP auditing system is utilized. The properties that we capture in our logic include invariant properties of a system, forensic hypotheses, and generic or specific attack signatures. Moreover, we discuss the admissibility of forensics hypotheses and the underlying verification issues.

Throughout our research we realized the significance the Windows registry can provide when correlated with the logs. The registry, being a source of system and application information, provides a reference point when detecting anomalies in the logs. Correlating the registry with the logs leverages the forensic analysis adding evidence to the investigation. We present the method of the correlation as well as a proof-of-concept implementation of the correlation of logs with the registry.

Acknowledgments

This dissertation could not have been written without the help and support of Dr. Mourad Debbabi and Dr. Amr Youssef who not only served as my supervisor but also encouraged and supported me throughout my academic program. They have been great teachers and managers during my Masters degree and I thank them for their endless patience, enlightening guidelines and unconditional support. I convey special acknowledgement to the Faculty of Engineering and Computer Science, Concordia Institute for Information Systems Engineering and the funders of the project namely Bell Canada and the Department of National Defense for providing the financial means and laboratory facilities. Finally, I take this opportunity to express my profound gratitude to my beloved parents for their moral support and patience during my study at Concordia University.

Contents

List of Figures

List of Tables

Chapter 1

Inroduction

Attacks on IT systems are increasing in number, and sophistication at an alarming rate. These systems now range from servers to mobile devices and the damage from such attacks is estimated in billions of dollars. However, due to the borderless nature of cyber attacks [9], many criminals and offenders have been able to evade responsibility due to the lack of supporting incriminating evidence. In this context, cyber forensics play a major role by providing scientifically proven methods to gather, process, interpret, and present digital evidence to bring a conclusive description of cyber crime activities. The development of forensics IT solutions for law enforcement has been limited. Although outstanding results have been achieved for forensically sound evidence gathering, little has been done on the automatic analysis of the acquired evidence. Furthermore, limited efforts have been made into formalizing the digital forensic science. In many cases, the forensic procedures employed are constructed in an ad-hoc manner which impedes the effectiveness and the integrity of the investigation.

One of the most common sources of evidence that an investigator should analyze is logged events from the activities of the system that are related to the incident in question. Indeed, having the logs from all system events during the incident will reduce the process of forensics analysis to event reconstruction. However, log analysis depends largely on the analyst's skills and experience to effectively decipher complex log patterns and determine what information is pertinent and useful to support the case at hand. Despite the paramount importance of this aspect, insufficient research effort has been dedicated to the automation of forensic log analysis. The main purpose of this thesis is to introduce a formal and automatic log analysis technique. The advocated approach caters for:

- Modeling of events and logical representation of properties that should be satisfied by the traces of logged events.

- Formal and automatic analysis of the logs identifying specific pattern of events or verifying a particular forensic hypothesis.

Further to the above mentioned, in our work we present two types of correlation:

1. correlation of events from different log files ("Event Correlation") and,
2. correlation of events from log files and Windows registry ("Event with Object Correlation").

Event correlation aims to identify relationships between events from different log files. If event x occurs in log file A and event y occurs in log file B, where x and y have a relationship r, then events x and y can be correlated and hence adding value to the investigation. We present event correlation between different log files through our proposed formal approach, whereby, a correlation rule can be modeled using a logical expression.

As for event with object correlation, the Windows registry contains a wealth of information relevant to a forensic investigation. Data obtained from the registry can be used to construct models of the system at hand. These models have a significant role in the detection of log file tampering. Furthermore, the models constructed from the registry can be correlated with events stored on a log file. One of the benefits of this correlation is elimination of doubtful and ambiguous sets of actions related to a malicious incident. The registry can be correlated with more then just log files, however, we limit the scope of our research to show the usefulness of such a correlation only.

The contribution of this thesis is the better use of evidence gathered during an investigation to prove or disprove facts, determine if the evidence is corrupted, and get the most out of the resources at hand. This is done through the proposition of two approaches for the forensic analysis of logs. The first being a formal approach to the problem of analyzing logs with the purpose of gathering forensic evidence. The second is the correlation of logs with the registry to provide further evidence that will leverage a forensic investigation. Besides the uniqueness of our two approaches, they enable the forensic analyst to establish reliable evidence to present in a court of law. The registry correlation approach can be used to gather system information such as: which logs are enabled and would be expected to be found, what services are installed along with their configuration, the user accounts on that machine and their access privileges, etc. This information gives the analyst the required system background and may be used to validate the authenticity of the logs. Using this information, the analyst can search for incidents related to the compromise of the system by constructing hypotheses using our formal approach or using the predefined rules. We also discuss the implementation of an automated system for correlation and analysis whereby the correlation rules are defined in XML and a verifier component which gets the rules as input from the XML parser and verifies if these rules are satisfied.

This thesis is organized into eight chapters. In Chapter 2, we discuss the approaches for log analysis including selected papers on cyber forensics. We also discuss an approach for the generalization of logs which we assessed to determine if it can be used to facilitate our analysis methodology. Chapter 3 provides the background required to construct windows related log models. We conducted an extensive analysis of the Windows logging system as well as Windows security related components in order to understand how processes work in a Windows operating system. Chapter 4 is devoted to the logic used to express properties of the model using the Windows logs, namely the security log, application log, system log and the Windows firewall log. In Chapter 5 we present our approach to the formal modeling of logs. Chapter 6 introduces a new concept which is correlating the Windows registry with the Windows logs. Chapter 7 is devoted to the implementation of a framework that can be used to parse the logs and the registry. Included in the framework is the log analysis using the formal approach discussed in Chapter 5, and the correlation with the registry approach discussed in Chapter 7. After the discussion about each implementation, we present a case study to further elaborate our proposed approaches. Finally, Chapter 8 brings us to the conclusion of the thesis where we also discuss possible improvements and future work.

Chapter 2

Log Analysis Approaches

Log analysis has been a topic dealt with over many years. The techniques used, the types of logs analyzed and the purpose of the analysis are all factors contributing to the wealth of work in this area. In this section, we present the different approaches used in log analysis. The different categories of log analysis we discuss are: deterministic log analysis, artificial intelligence, state-based analysis, statistical log analysis, temporal sequence learning, and log correlation. As this thesis deals with cyber forensics it is beneficial that we present some related cyber forensic approaches which we came across during the course of our research. Finally, in this chapter we present formal approaches in log analysis as well as generalization of the logs.

2.1 Deterministic Log Analysis

Deterministic log analysis is based on the fact that events or sequence of events have inevitable consequence of antecedent states. The correct next step depends only on the current state. This contrasts with an algorithm involving backtracking where at each point there may be several possible actions and no way to chose between them except by trying each one and backtrack if it fails.

General and Systematic Methods for Intrusion Detection

The authors in [10] have developed some general and systematic methods for intrusion detection. They have built a framework using data mining techniques to discover consistent and useful patterns of system features that describe program and user behavior. To detect anomalies and known intrusions, they have used a set of relevant system features to compute (with inductively learned) classifiers. The two major data mining algorithms they have implemented are: the *association rules* algorithm and the *frequent episodes* algorithm, which are used to recognize intra- and inter-audit record patterns. To meet the challenges of both efficient learning (mining) and real-time detection, they have proposed an agent-based architecture in which the learning agents continuously compute and provide the updated models, while a detection agent is equipped with a learned and periodically updated rule set from the remote learning agent. This research is still at its initial stage. Implementation and validation of the methodology details still need to be done and hence not much can be determined in terms of solid opinions at this point.

Rule-Based Audit Trail Analysis

Mounji et al [11] developed a system for on-line analysis of multiple distributed data streams. The system uses the rule-based language from the ASAX project [12] to filter audit data at each monitored host and to analyze filtered data gathered at a central host. The motivation for using the ASAX project to develop a distributed audit trail analysis system is mainly due to the advantages of correlation. The correlation of user actions at different hosts could reveal a malicious behavior while the same actions may seem legitimate if considered at a single host. The advantages of using ASAX, according to the authors, are universality, the RUSSEL language, and efficiency. Universality is achieved through Normalized Audit Data Format, NADF, which allows straightforward translation of native files and fast processing of NADF records by the universal evaluator. Furthermore, the RUSSEL language "is a novel language specifically tailored to the problem of searching arbitrary patterns of records in sequential files" [11]. Furthermore, RUSSEL's built in C-routines makes extending the language quite easy. Lastly, the ASAX project is efficient due to the efficiency of RUSSEL. The operational semantics of RUSSEL exhibits a bottom-up approach in constructing searched record patterns. Furthermore, optimization issues are carefully addressed in RUSSEL. Although RUSSEL provides efficient analysis, the language itself has usability issues. The granularity of rules defined in RUSSEL is limited in comparison to other approaches which we will discuss later.

RUSSEL, being a rule-based language designed for analysis of sequential files in one pass, has been used in numerous projects, other then the above mentioned, supporting rule-based analysis. For example, in [12], the authors provided a brief survey of problems related to audit trail analysis as well as some approaches to deal with these problems. The authors then present the ASAX project which is supposed to deal with these problems providing an advanced tool to support rule-based analysis using RUSSEL. The discussed problems are: the disparity of security breach scenarios, the huge amount of data involved in audit trails, reusability of audit trail and analysis tools, and finally the conviviality of the interface supplied to the security manager. In the proposed solution, the ASAX project, a standard format for audit trails is developed. An audit trail is transformed to the NADF by format adaptors. The query language developed using RUSSEL is not meant to be used by security officers until a higher level query language, RUSSEL2, is developed. The disadvantage of RUSSEL2 is that predefined rules using RUSSEL have to be developed and stored in a database for use during the analysis. The type of rules generated by the security officers using RUSSEL2 is determined by the predefined rules stored in the database. This places a limitation to creating customized rules which may be case specific.

Another research in the line of rule-based analysis which we found interesting is P-BEST. P-BEST is an expert system development toolset called the Production-Based Expert System Toolset. The work in [13] describes P-BEST and how it is employed in the development of a modern generic signature-analysis engine for computer and network misuse detection. They present rule sets for detecting subversion methods against which there are few defenses (specifically, SYN flooding and buffer over-runs) and provide performance measurements. They claim that the simplicity of the P-BEST language and its close integration with the C programming language makes it easy to use while still being very powerful and flexible.

2.2 Artificial Intelligence

Artificial Intelligence (AI) can be seen as an abstract agent with functional intelligence. AI is used in cases where human intelligence would be required to solve a problem. In log analysis, AI has been used to simulate the analysis that would be required by a person, but in a more structured manner. In this section, we present some of the approaches used and evaluate a predominant few.

SmartSifter

The SmartSifter [14] is an outlier detection system based on unsupervised learning of the underlying mechanism for data generation. By using outlier, which is a fundamental issue in data mining, Smartsifter can be used for different purposes such as fraud detection, network intrusion detection and network monitoring. The mechanism is based on a probabilistic model which uses a finite mixture model. Each new input datum is examined to see how much it has deviated from a normal pattern. At the same time, an on-line learning algorithm is employed to update the model. The datum is given a score showing how many changes have happened after learning. A high score means that the datum is an outlier. The work presented in [14] distinguishes itself from other such tools in that it is on-line, meaning the detection is in real-time, in contrast to other work on outlier detection in statistics and data mining which use batch-detection processing. The approach used by the authors of [14] is that every time a datum is input, SmartSifter employs an on-line learning algorithm to update the model. Then SmartSifter scores the input datum based on the learned model which indicates how much the model has changed. A high score would indicate a high possibility that the datum is an outlier. Given the usability of SmartSifter for intrusion detection, its success was more in the area of sorting through large matrices.

Artificial Anomalies

In [15], the authors have proposed an algorithm to generate artificial anomalies to force the inductive learner to find out a more accurate boundary between known classes (normal connections and known intrusions) and anomalies. Their experiment on the KDD99 [16] data set shows that the model is capable of detecting more than 77% of all unknown intrusion classes with over 50% accuracy per intrusion class. However, the way to generate anomalies is not clear.

Log Analysis-Based Intrusion Detection via Unsupervised Learning

In [17], an intelligent log analyzer that can detect known and unknown network intrusions was developed. The log analyzer uses a data mining framework and is trained with unsupervised learning algorithms, the k-means algorithm and Autoclass. The author's proposed approach using unsupervised learning and then capture via supervised labelling is claimed to be working with the help from some pre-labelled data, assuming that those pre-labelled data are representative enough to cover clusters with intrusions. This method is suitable for detecting intrusions data with limited labelled data. It may not be able to detect those unknown attacks if the clusters containing unknown attacks are not marked out by the given labelled data.

Features and Models for Intrusion Detection

The main contribution of [18] is an automatic feature creation scheme, where feature extractors are created by using various datamining techniques. After the extraction process, it uses machine learning algorithms to the processed audit records to generate intrusion detection rules. It looks at the IDS from the network layer and all the way up to the application layer. The approach used here significantly reduces the manual analysis and encoding of intrusion patterns. Furthermore, the fact that the resultant models are computed and validated using large amount of audit data makes them more effective.

Neural Networks

In [19] a new way of applying neural networks to detect intrusions is proposed. A user leaves a print when using the system; a neural network can be used to learn this print and identify each user much like detectives use thumb prints to place people at crime scenes. If a user's behavior does not match his print, the system administrator can be alerted of a possible security breach. A back propagation neural network called NNID (Neural Network Intrusion Detector) was trained in the identification task and tested experimentally on a system of 10 users. The problem with this approach is that with the increase in the number of users, the number of false alarm increases. Thus as the number of users increases it will take more time to train the network and a larger network will be required.

In [20] a machine learning approach is utilized to reconstruct a post event timeline. This timeline will be applied during a forensic investigation as a method of presenting evidence, as well as being provided as evidence in and of itself. This is accomplished by monitoring file system accesses, taking file system snapshots at discrete time intervals, and using this data to generate a recurrent neural network to recognize execution patterns of individual applications. Using machine learning to close gaps generated temporary information timeouts reflecting in an incomplete timeline is a useful technique. However, training separate neural networks is required for different applications and a new set of clean-state instances of file systems manipulation would be required for newer versions of applications.

Multiple Self-Organizing Maps for Intrusion Detection

Neural networks capable of unsupervised learning can provide a powerful supplement to current intrusion detection techniques [21]. After learning the characteristics of normal traffic or user behavior, these networks can identify abnormalities without relying on expectations of what abuse will look like. [21] analyzes the potential of the Kohonen self-organizing map, which is a powerful mechanism for automatic mathematical characterization of acceptable system activity, to narrow the intrusive behaviors that would not be caught by a detection system. The advantage of this approach is by using the self-organizing map whereby intrusion behaviors don't need to be specified. This is especially useful for zero-day attacks when it comes to intrusion detection. Considering this for forensic log analysis is only useful where only an automated analysis is required. Hypothesis testing is not possible with this approach.

Agent-Based

In [22], the authors present the frameworks of distributed agent-based real time network intrusion forensics system. The framework dumps the misbehavior packets traffic based on filtering rules, analyzes the overall log data and traffic data to discover the potential misbehavior, and launches the investigation at the intrusion time. They also addresse some novel approaches for network forensics such as: network forensics server, network forensics database, network forensics agents, and real time network investigation. Network forensics server integrates the forensics data and analyzes them, manages the network packet filter and captures behaviors on the network monitor, and launches the investigation program. Network monitor captures network traffic. Network investigator provides mapping topology data, actively investigates a target when prompted by the server, and can launch the real time investigation in response to a network intrusion. Distributed agents gather evidence from the distributed hosts. The evidence includes IDS logs, system logs, and sensitive or critical file signature on the hosts. Through this framework, when an intrusion is detected, the forensics server starts a "forensic investigation" to determine malicious activity rising from the intrusion through data mining analysis.

Anomaly Detection

The theorem proposed in [23] guarantees that the audit trail has repeating patterns. Based on fuzzy-rough set theory, hidden fuzzy relationships in audit data are uncovered. The information about the repeating data and fuzzy relationships reflect patterns of users' habits. The author presents two types of soft patterns: repeating records and fuzzy relationships between data. Since audit data can be interpreted as an infinite input stream of records of a database, repeating patterns are guaranteed. Furthermore, the fuzzy-rough set methodology is used to prove the repeating fuzzy relationships in the audit trails. Based on these findings, the author proves that rules can be extracted from audit trails. However, besides mathematical examples, no real world examples are provided to illustrate the theories.

2.3 State-Based Analysis

STATL [24] is an extensible state/transition-based attack description language designed to support intrusion detection. The language allows one to describe computer penetrations as sequences of actions that an attacker performs to compromise a computer system. A STATL description of an attack scenario can be used by an intrusion detection system to analyze a stream of events and detect possible ongoing intrusions. It defines domain-independent features of attack scenarios and provides constructs for extending the language to describe attacks in particular domains and environments. The STATL language has been successfully used in describing both network-based and host-based attacks, and it has been tailored to very different environments, e.g., Sun Microsystems Solaris and Microsofts Windows NT. An implementation of the runtime support for the STATL language has been developed and a toolset of intrusion detection systems based on STATL has been implemented. The toolset was used in a recent intrusion detection evaluation effort, delivering very favorable results.

2.4 Temporal Sequence Learning

Intrusion detection through system call traces was discussed in [25]. Unusual behavior within a computer system can be detected by monitoring the system calls being executed by a program. Depending on the auditing used, data recorded in logs can encompass the usage of system recourses and system calls made by some collection of processes. To study the structure of normal traces and determine how the structure is violated during an intrusion a deterministic finite automata (DFA) was used based on the repetitive long system calls in the sendmail log.

The work in [26] presents an approach on the basis of instance-based learning (IBL) techniques. In order to transform the anomaly-detection task in an IBL framework, the authors employed an approach that transforms temporal sequences of discrete, unordered observations into a metric space via a similarity measure that encodes intra-attribute dependencies. Classification boundaries were selected from an a posteriori characterization of valid user behaviors, coupled with a domain heuristic. An empirical evaluation of the approach on user command data demonstrated that they can accurately differentiate the profiled user from alternative users when the available features encode sufficient information. Furthermore, the authors demonstrated that the system detects anomalous conditions quickly, which is an important quality for reducing potential damage by a malicious user. Thus they present several techniques for reducing data storage requirements of the user profile, including instance-selection methods and clustering.

2.5 Statistical Analysis

P. Helman et al. [27] model computer transactions as generated by two stationary stochastic processes, the legitimate (normal) process N and the misuse process M. They demonstrate that the accuracy of misuse detectors is bounded by a function of the difference of densities of the process of N and M over the space of transactions.

In [28], the authors discuss SRI International's real-time intrusion-detection expert system (IDES) system which contains a statistical subsystem that observes behavior on a monitored computer system and adaptively learns what is normal for individual users and groups. The statistical subsystem also monitors observed behavior and identifies behavior as a potential intrusion (or misuse by authorized users) if it deviates significantly from expected behavior. The multivariate methods used to profile normal behavior and identify deviations from expected behavior are explained in details. The statistical test for abnormality contains a number of parameters that must be initialized and the substantive issues related to setting those parameter values are discussed.

The authors in [29] discuss Wisdom and Sense (W&S), a computer security anomaly detection system. W&S is statistically based. It automatically generates rules from historical data and, in terms of those rules, identifies computer transactions that are at variance with historically established usage patterns. Issues addressed include how W&S generates rules from a necessarily small sample of all possible transactions, how W&S deals with inherently categorical data, and how W&S assists system security officers in their review of audit logs. Preliminary results with W&S show that the software does periodically detect anomalies of high interest even in data though to be free of such events.

Obstacles to achieving anomaly detection in real time include the large volume of data associated with user behavior and the nature of that data. The work in [30] describes preliminary results from a research project which is developing a new approach to handling such data. The approach involves nonparametric statistical methods which permits considerable data compression and which supports pattern recognition techniques for identifying user behavior. This approach applies these methods to a combination of measurements of resource usage and structural information about the behavior of processes. Preliminary results indicate that both accuracy and real time response can be achieved using these methods.

In [31], anomaly and misuse detection approaches were developed and applied to the Basic Security Model (BSM) of Suns Solaris operating environment. The anomaly detection approach uses the statistical likelihood analysis of system calls, while the misuse detection approach uses a neural network which is trained on groupings of system calls. Needless to say, each of anomaly and misuse detection have their inadequacies. The aim of this research is to combine both approaches to overcome the drawbacks of each method. By combining the end results of these two methods, the false negative and false positive rates are improved.

To get the desired result for anomaly detection, training data had to be produced. A program called praudit was used to translate the BSM audit data binary file into a readable format for parsing. A Perl script then separated audit signals of different users into respective files. For each user file, the entire sequence of audit events was converted into a file of correlated numbers that represent signal events. Once the time-stamped signal files have been created for each of the users, the training data in the form of feature vectors with all the possible signals is then produced. The entire sequence of signals is stored in an array fashion. Given the entire sequence of signal numbers for a user, a sliding window was created to determine how many signals to consider in one pattern. Through tests, it is shown that the optimal sequence window lengths vary per user and the respective normal profile size. As for the misuse detection, instead of learning the behavior of users over time, this method learns an entire event and creates a feature vector from this. For example, instead of the sliding window used in the anomaly detection approach, a sequence of signals was grouped into an event. An event is used to mean the collection of signals associated with a particular command or action. To generate the events, the collected signals of both user and system level events from the same audit source as the training source are used. Feature vectors were generated as before using all possible signals. An event is extracted from the BSM audit log by combining all signals within several microseconds that have the same audit session id. Each event pattern was then classified as normal or abnormal system behavior with a 0 or 1 respectively. The results of the testing showed that combining anomaly and misuse detection drastically reduced false negative errors. The output of the detection models shows overlapping in the graphed results. Merging the several outputs provided by this approach better shows areas that need human audit to determine if a real attack has taken place.

This approach introduces an interesting idea that may be worth investigating for the sake of forensics. Using misuse and anomaly detection approaches to reduce the false negatives and better represent the data statistics to narrow down the possible occurrences of an incident is an innovative approach. The problem would remain to train the detection engines and provide the proper test data. If the test data is flawed, this will affect all the analysis.

2.6 Log Correlation

There has been a considerable number of approaches on log analysis and correlation, though the direction was in the interest of intrusion detection rather than forensic log analysis. The difference is that intrusion detection aims to detect events that indicate an attack has been made or is in progress. Forensic log analysis on the other hand would look for events that led to the attack as well as events related to the consequences of the attack. Intrusion detection approaches for log analysis and correlation can be extended for the purpose of digital forensics, but the approach "as-is" is not sufficient.

Through our research, we came across a correlation approach based on triggering events and common resources [32]. The concept used is to partition sets of alerts into different clusters such that alerts in the same cluster may correspond to the same attack. Each set of attacks are consistent with relevant network and host configurations. The correlation of alerts is done in three stages. In the first stage, events observed by security systems that trigger an alert is gathered, these are called triggering events. The type of events that are focused on are low-level events such as TCP connections. In the second stage, alerts are examined to determine their consistency with relevant network and host configurations. In the last stage, attack scenarios are built through the input and output resources. Input resources being the events and the output resources being the alerts. Informally, input resources are the necessary resources for an attack to succeed, and output resources are the resources an attack can supply if successful.

An interesting concept mentioned in [32] is inference between events: two events are similar if they have the same event type, attribute name, and values. However, considering the existence of implication relationships between events (the occurrence of one event implies the occurrence of another event), we realize that the concept of similarity can be extended beyond this intuition to accommodate event implication. Besides their concept of inference, they extend their correlation by identifying casual relationships between attacks through discovering common resources between input and output resources. If one attack's input resources include one resource in another attack's input resources, they can correlate these two attacks together.

The disadvantage of the approaches discussed above is that it does not support an automated procedure. The analysis of events and alerts to construct attack signatures and correlation is a very tedious task especially since the logs of IDSs can grow tremendously in size and filtering out the unrelated information and all the false positives is quite time consuming. Although constructing the signatures is a one time task, updating the signature database will require the same task again.

In terms of other approaches in log correlation, research on alert correlation can be classified into four categories based on [32]:

- Similarity based approaches [33, 34, 35, 36]. These approaches group alerts based on the similarity between alert attributes, i.e. they are essentially clustering analysis. Other techniques also include alert clustering, which uses a novel similarity measure: triggering events. Triggering events are similar to root cause [34] concept in that they represent the reason why the alerts are flagged. However, triggering events focus on low-level events (though high-level events are possible). Furthermore, an assumption is made which states that security systems or domain knowledge can tell us triggering event types for each alert type, while root cause analysis concentrates on high-level events and clustering techniques are used to discover root causes. In [33], Cuppens proposes to use alert clustering to identify "the same attack occurrence", where expert rules are used to specify the similarity requirement between alerts.
- Predefined attack scenario based approaches [37, 38], which detect attacks according to well defined attack scenarios. However, they cannot discover novel attack scenarios.
- Pre/post condition based approaches [39, 40, 41]. Through (partially) matching the post-condition of one attack with the pre-condition of another, these approaches can discover novel attack scenarios. However, specifying pre-conditions and post-conditions for each attack is time-consuming and error-prone. Our techniques on building attack scenarios fall into this pre/post-condition based category. However, since our approach uses resources to specify pre/post-conditions, compared with the predicate based specification [39, 40], it is easy to specify and partially match input and output resources, and easy to accommodate new attacks.
- The multiple information sources based approaches [42, 43, 44] that are concerned with distributed attack discovery. The mission-impact-based approach [43] ranks the alerts based on the overall impact to the mission of the networks. The M2D2 approach [42] proposes a formal model to describe the concepts and relations about various security systems. The DOMINO approach [44] is a distributed intrusion detection architecture targeting at coordinated attack detection with potentially less false positives. We consider these techniques as complementary to ours.

2.7 Cyber Forensics

The main purpose of our log analysis is for cyber forensics, therefore previous work on cyber forensic analysis coincides with our research. Digital forensic analysis is still at its infancy therefore not much research had been dedicated to forensics directly. In the following section we discuss some selected approaches used for analysis which can be comparable to our intended purpose.

The work presented in [45] proposes an approach to post-incident root cause analysis of digital incidents through a separation of the information system into different security domains and modeling the transactions between these domains. The proposed approach is best suited for large and complex investigations. To justify their claim, the authors present a case study of a SQLSlammer worm infection on a large multinational enterprize. The enterprize network was designed for high availability with little consideration of security. The worm could have infected the internal network through different entry point. The aim of the investigation was to determine the exact entry point and apply appropriate countermeasures. Based on the Digital Forensic Research Workshop (DFRWS) investigative framework, the authors developed an End-to-End Digital Investigation (EEDI) approach supported by a Digital Investigation Process Language (DIPL). The components of the investigative process are as follows:

- Problem Statement - this is a component required to have a structured and formal investigation process.
- The DFRWS Framework - the framework defines the actions or "elements" necessary in each stage or "class" in an investigation.
- The End-to-End Investigation Process - a collection of generalized steps to be taken in conjunction with the DFRWS framework. Some of the important steps it addresses are event normalization and de-confliction as well as two levels of correlation. One level prior to event normalization and the second level considers both normalized and non-normalized events. Lastly the corroboration step which considers only non-normalized events.
- The Digital Investigation Process Language [46] - the purpose of which is to allow a structured description of the investigative process. It provides semi-formal description using Semantic Identifiers vocabulary arranged in s-expressions to describe tasks, functions and actions.
- Colored Petri Net Modelling (CPN) [47] - this leads to a formal model and in some cases possible outcomes. The CPN modeler has been used for this purpose. The modeler is used for two purposes. The first purpose is to create a model of the DIPL investigation and output probable outcomes of a set of investigative actions. The second purpose is to validate the DIPL.

This approach is very structured and a good part of the investigation is automated and hence facilitates the work of the investigator. In an ideal scenario, this approach is very effective constructing the DIPL process given that the evidence is complete. However, in most cases, the evidence is not complete and the gathered actions might not be complete to construct a DIPL process so the modeler will fail to provide its outcome and will fail to validate the DIPL process.

The timeline analysis approach [48] consists of analyzing digital evidence from multiple sources such as the computer system, network, and peripheral devices to develop a timeline of the events that led to the incident. The work in this paper provides a process for extracting the digital evidence and correlating it with external evidence such as phone records, testimonies, and physical evidence to construct a timeline of the events. This timeline is a historical road map that provides detailed information relevant to the investigation. They begin by determining what evidence could be used to extract reliable evidence for the purpose of timelining. This reliable evidence is determined to be the following: files and documents, e-mail, login/logout events, and web access. The next step is to correlate the gathered evidence along with whatever physical evidence is available. When correlating different types of evidence the issue of different time zones arises. Specific attention should be directed to this issue to ensure that the timeline reflect the correct sequence of events.

The process discussed is determined to be a brief introduction to timeline analysis. Besides the issue of different time zones, there is the issue of different time stamps. Even within logs, each log type may use a different time and date format. The fact is, if the digital evidence is complete and not tampered with, timeline analysis will provide the exact historical road map of events that led to a certain incident as well as the actions taken after the incident. Furthermore, correlating this timeline analysis with physical evidence will recreate an undebatable sequence of events pertaining to the crime. Needless to say, more work needs to be invested in this area.

With the increased size and complexity of digital evidence to analyze and with the increased complexity of attacks, an automated expert system is required, as stated in [49]. With the use of an automated expert system, the reliance on technical expertise is reduced. Rather than having an opinion based on a person's best effort and limited recourses, an automated analysis is used where the reasoning process is fully documented, transparent and deterministic. The approach presented in [49] is an expert system with a decision tree that uses predetermined invariant relationships between redundant digital objects to detect semantic incongruities. The approach is based on the fact that even if an attacker tries to modify the system to conceal evidence of the attack, side effects of his presence may still exist. Only the perfect attacker will be able to flawlessly recreate the state of the system as it was before her violation [49].

In the proposed approach, the automated system first finds data objects with redundancies that must exist in a secure system. Once this is done, the next step is to search through evidence collected from the compromised system for contradictions. Once raw data is collected and preserved, it must be aggregated in a rational way into abstract objects. Operating systems and their applications build digital objects using lower level abstraction. Some objects may be duplicated to increase performance or for security purposes. The authors noted different examples of ways used by the system to detect incongruities and anomalies within the digital evidence. However, the advantage of their system is the ability to learn invariant relationships of custom software which is harder for an attacker to know about. If an attacker modifies data related to the custom software chances, are that she will not be able to modify all the data due to insufficient knowledge of its whereabouts or even existence. Based on these contradictions and readily available information it is possible to identify reasonable explanations or evidence of malicious activity. This approach shows the possibility to answer useful forensic questions by using data from common mechanisms that may not have been intended for security purposes. It is possible to do so with no preparation before an attack. This post-fact analysis is an extensible approach that can be evolved to include more semantic data relationships specific to host, network and application abstraction layers. In addition, these relationships may be detected and verified in an automated fashion in systems that have functional redundancies. Although it is possible that the minimum amount of information may not exist to completely solve the crime, an imperfect attacker will leave clues.

Baseline analysis is another approach used for cyber forensic investigations. A study of system baselining was presented in [50] where the authors discuss different approaches to demonstrate the utility of baseling. The authors also present FTime, a system baselining and evidence collection tool which is well suited to handle the types of applications and environments incident handlers are likely to encounter. The work presented in this paper gives a very comprehensive idea about the concept of baselining. The authors have defined baselining terminology, explained mechanics of baselining, and compares different baselining techniques.

The main idea of baselining is to capture a snapshot of a system to create a baseline, which is a set of critical observations or data used for comparison. There are four factors involved in baselining: provenance, perspective, acuity, and integrity. Provenance is the origin or source of an object. One way to determine the provenance of a given object is to compare its hash to a previously recorded hash of the object in question. Perspective is the capability of vieing objects in their true relations or relative importance. Acuity is keenness of perception, the relative ability of a tool to resolve detail. Last but not least, integrity which is an unimpaired condition or the quality or state of being complete or undivided. The integrity of a baselining tool along with its input and output must be protected throughout the baselining process to ensure the reliability of the baseline.

As far as baslining techniques, we will just mention them briefly without going into much details. The are four different techniques used for baselining are: alternate platform, alternate operating system, single-user mode, and multi-user mode. Alternate platform involves using a dedicated stand-alone system which has been configured specifically to mount and scan disks extracted from subject systems. Alternate operating system whereby the subject system is mounted into an alternating operating system that can access and scan subject file systems. Single-user-mode is a simpler technique used where the single-user mode of the subject system is used. This technique can not be used on all system, depending if the operating system supports such a mode. Multi-user-mode on the other hand is when a baseline is created under normal system operation. This technique is the least sound, yet due to operational constraints, it may be the only possible technique to create a baseline of the subject system.

2.8 Cyber Forensic Formalization

Our main concern is formalization of forensic log analysis. To this date there has not been much effort invested in this area. Even the work on formal digital forensics is still in its early stages. One of the most significant and most related research in this area is presented in [51], where Pavel Gladyshev proposed a formalization of digital evidence and event reconstruction based on finite state machines. In his work, the behavior of the system is modeled as a state machine and a recursive model-checking procedure is proposed to verify the admissibility of a forensic hypothesis. However, in the real world, modeling the behavior of a complex system such as an operating system as a state machine diagram is burdensome and sometimes impossible to achieve because of complexity issues. Other research on formalized forensic analysis include the formalization of event time binding in digital investigation [52, 53], which proposes an approach to constructing formalized forensic procedures. Nevertheless, the science of digital forensics still lacks a formalized approach to log analysis.

Formal log analysis and hypothesis verification is of paramount importance to forensic science. As an example, invariant properties of the system cannot be modeled and analyzed through most of the approaches mentioned earlier in this chapter. The absence of a satisfactory and a general methodology for forensic log analysis has resulted in ad-hoc analysis techniques such as the log analysis presented in [54] and operating system-specific analysis discussed in [55]. In what follows, we discuss the more prominent approaches of formal log analysis that we came across while conducting our research.

Roger et al. [56] exploit the idea that signatures are best expressed in a logical language including temporal connectives to express ordering of events. Roger et al. propose a logic consisting of flat, Wolperstyle linear-time formula. They demonstrated the possibility of on-line detection of complex correlation of events using a declarative style of signatures, expressed in a suitable variant of temporal logic. In their approach, the authors interpret the logs as Kripke models. They show that the classic linear-time logic proposed in [57] does not fit their needs suggesting that a propositional temporal logic is insufficient. The temporal logic should have first order variables to be able to check that a user X copying a file, for example, is the same one that set the file's setuid bit. Their optimized implementation had successful results in terms of performance. However, this is an exponential-time algorithm, as the log file size increases, the time to analyze it increases exponentially. Nonetheless, detection of complex correlations of events is feasible and comparable to simpler intrusion detection systems that only do filtering and counting. As far as usability is concerned, generating complex signatures causes combinatorial explosions if an error was made while writing the signatures. Thus a signature should be accurate no matter how complex it is, otherwise the system will crash.

In [58], a temporal logic approach was adopted for signature-based intrusion detection. Intrusion patterns were specified as formulas in expressively rich and efficient moniterable logic. EAGLE, the logic utilized, supports data-values and parameterized recursive equations and allows the expression of security attacks with complex temporal event patterns as well as attack signatures which are inherently statistical in nature. The proposed approach is implemented in a prototype called MONID which detects intrusions either online or offline. The design of the framework is as follows: information about system-level events are sent to the server. These events are obtained either from offline log files or generated online by appropriately instrumented application code. The server merges the events from various sources by timestamp and preprocesses them into an abstract intermediate form to generate a single event trace. The monitor subsequently monitors this event trace against a given specification and raises an intrusion alert if the specification is violated. The propositions or expressions of an EAGLE formula are assumed to be specified with respect to the fields of the event record. At every event the algorithm evaluates the monitored formula on the event and generates another formula. At the end of the event sequence, the value of the evolved formula is determined.

If the value is true the formula is satisfied by the event sequence, otherwise the formula is violated. It should be noted that the logic EAGLE has a finite trace semantics. The monitoring algorithm can determine the satisfaction or the violation of a formula at the end of a trace only. However, in intrusion detection, end of trace makes no sense as the sequence of events can be theoretically infinite. In that situation, an alarm has to be triggered as soon as a property is violated. This is done by checking after every event if the formula becomes satisfiable, rather than waiting for the end of the trace. Still this technique does not seem too promising when dealing with large traffic. The offline monitor would be pretty efficient since there is no real-time analysis involved which might cause the monitor to crash due to any error. The logic, although not too elegant, can be used to model signatures, but a smarter monitor would be required for heavier traffic of events since the current monitor has a finite trace semantics.

A logic more expressive than [58] was used in the works of Goubault et al. [56]. The proposed logic allows attack signatures involving real time constraints and statistical properties to be expressed. The idea was to specify the intended behavior of security-critical programs as temporal formulas involving statistical predicates, and monitor the system execution to check if it violates the formula. By so doing, attacks can be detected even if they are previously unknown. For basic formulas, this proposed logic is easy. However, as the formula becomes more complicated, representing them in the logic represents a challenge for some users who have no background in logic.

The use of Interval Temporal Logic (ITL) in modeling temporal properties of multi-event attack signatures was presented in [59]. Misuse detection rather than anomaly detection approach was used for intrusion detection where attacks were assumed to be well-known sequences of actions. The proposed model is based on a high-level declarative Interval Temporal Logic (ITL) which is an expressive and well-studied formalism that is successfully applied to different areas of specification and verification of real-time systems. Moreover, ITL provides a straightforward method to define complex temporal features and has been used formerly in attack specifications. The use of a temporal logic introduced operators such as 'always', 'sometime', and 'at the next moment'. In the context of misuse detection, time is considered as a discrete and linear structure. Thus by using ITL, the authors were able to use the notion of satisfaction by an interval which is a finite discrete linear structure.

The proposed model aims to support the task of modeling time relations between events in multi-event attack signatures. However, the work in this paper relies on the fact that interval temporal logic formalism has already been proven to be a convenient and expressive modeling tool. Thus they do not discuss any proof of concept implementation to demonstrate performance, efficiency, and accuracy nor do they present a case study to elaborate on the use of their proposed method.

2.9 Generalization of Log Events: The Common Base Event

In an attempt to have a generic log event in order to facilitate the correlation, IBM has come up with the Common Base Event [60]. The purpose of the Common Base Event is to facilitate intercommunication between different enterprize components that support logging. It is very efficient in regards to monitoring, tracing, and managing performance of different components. By creating a generic event format, all the component logs can be translated to this format which provides ease of manipulation and analysis. In addition to generating a common format for logs, IBM has created APIs and tools to parse, correlate, and analyze common base events.

The Common Base Event tool has six components that are responsible for reading and parsing a raw log format:

- Adapter - this is a configuration file used to parse and convert log files to the Common Base Event. Each configuration file contains one or more context, and each context is related to a specific log file. The context describes ordered components which are used for log file processing.
- Sensor - reads the content of the log file for processing.
- Extractor - the sensor outputs a collection of lines to the extractor, which separates them into message boundaries. By default, the extractor class uses regular expressions to parse each line, however, the extractor class can be changed or customized as needed.
- Parser - the role of the parser is to take the messages that have been separated by the extractor and maps string values to the Common Base Event attributes.
- Formatter - takes the output provided by the parser and builds the correct Java object instance.
- Outputter - writes Common Base Event records provided by the formatter.

Once we have the Common Base Event, we can now easily correlate, log type and analyze them, all by the use of APIs and tools provided by IBM. As mentioned earlier, the Common Base Event is ideal for monitoring, profiling, and testing in the context of performance. We tried to incorporate it for forensic analysis, however the structure of the event as modeled by IBM provides little significance for forensics. Fields that are important for a forensic investigator might be of no meaning for someone trying to monitor performance, and vice versa. We considered remodeling the Common Base Event for the use of forensics, but found that to do so, we will need to specify a generic event for different categories of logs. An IDS log event is very different from a system log event. We did not further investigate research in this area since this is out of the scope of this thesis.

Chapter 3

Windows Logging System

The Windows Event Logging service [61] enables an application to process, publish, and access events. The events are stored in event logs and can be checked by an administrator or monitoring tool to detect occurrences or problems on a computer and to monitor performance and accesses.

Many applications log events in various proprietary logs. These proprietary logs have different formats and display different user interfaces, mainly depending on the purpose of the log. Moreover, you cannot merge the data to provide a complete report. Therefore, you need to check a variety of sources to diagnose problems. Windows event logging provides a standard, centralized way for applications (and the operating system) to record important software and hardware events. The event-logging service stores events from various sources in a single collection called an event log. Furthermore, Event Tracing for Windows (ETW) provides APIs for application programmers to instrument event auditing controls for an application.

In the following sections of this chapter we discuss the details of the Windows logging system and the events generated. Specifically, we discuss the event tracing sessions, event types and event logging elements to understand the nature of events in a Windows environment. Then we discuss the event logging model which specifies how events are read and written. Finally, we list the types of logs that can be found in a Windows operating system emphasizing on the security log.

3.1 Event Tracing Sessions

A controller defines the session, which typically includes specifying the session and log file name, type of log file to use, and the resolution of the time stamp used to record the events. Event tracing sessions record events through the controllers from one or more providers. The session is

Event type	Description
Error	An event that indicates a significant problem. For example, if a service fails to load during startup, an Error event is logged.
Warning	An event that is not necessarily significant, but may indicate a possible future problem. For example, when disk space is low. If an application can recover from an event without loss of functionality or data, it can generally classify the event as a warning event.
Information	An event that describes the successful operation of an application, driver, or service. For example, when a network driver loads successfully.
Success Audit	An event that records an audited security access attempt that is successful. For example, a user's successful attempt to log on to the system.
Failure Audit	An event that records an audited security access attempt that fails. For example, if a user tries to access a network drive and fails.

Table 1: Event Table [3]

also responsible for managing and flushing the buffers.

Event Tracing supports a maximum of 64 event tracing sessions executing simultaneously. The 64 tracing sessions are divided into special and general purposes. There are two special purpose sessions and the rest of the sessions are available for general use. The special purpose sessions are:

- Global Logger Session - records events that occur early in the operating system boot process, such as those generated by device drivers.
- NT Kernel Logger Session - records predefined system events generated by the operating system, for example, disk IO or page fault events.

3.2 Event Types

There are five types of events that can be logged. All event classifications/types have well-defined common data and can optionally include event-specific data. The application indicates the event type when it reports an event. Table 1 describes the event types used in event logging.

3.3 Event Logging Elements

The following are the major elements used in event logging: Eventlog key, Event sources, Event categories, Event identifiers, Message files, Event log records, and Event data [4]. These elements are elaborated in the following sub-sections:

3.3.1 Eventlog Key

The Eventlog registry key contains several sub-keys called logs. Each log subkey contains information that the event logging service uses to locate resources when an application writes to and reads from the event log.

The event log contains three standard logs (Application, Security, and System), as well as custom logs. The structure of the Eventlog key is as follows:

```
HKEY_LOCAL_MACHINE
        SYSTEM
            CurrentControlSet
                Services
                    Eventlog
                        Application
                        Security
                        System
                        CustomLog
```

3.3.2 Event Sources

Each log key contains subkeys called event sources, which is the name of the software that logs the event. It is often the name of the application, or the name of a subcomponent of the application if the application is large.

The Security log is for system use only. Device drivers should add their names to the System log. Applications and services add their names to the Application log, or create a custom log. The

Registry Value	Description
CategoryCount	Number of event categories supported.
CategoryMessageFile	Path for the category message file. A category message file contains language-dependent strings that describe the categories.
EventMessageFile	Path for the event message file. Multiple files listed are separated by semicolons.
ParameterMessageFile	Path for the parameter message file. A parameter message file contains language-independent strings that are to be inserted into the event description strings.
TypesSupported	Can be one or more of the following values:
	EVENTLOG_AUDIT_FAILURE
	EVENTLOG_AUDIT_SUCCESS
	EVENTLOG_ERROR_TYPE
	EVENTLOG_INFORMATION_TYPE
	EVENTLOG_WARNING_TYPE

Table 2: Contents of the Message File [3]

structure of the event sources is as follows:

- *HKEY_LOCAL_MACHINE*
  - *SYSTEM*
    - *CurrentControlSet*
      - *Services*
        - *EventLog*
          - *Application*
            - *AppName*
          - *Security*
          - *System*
            - *DriverName*
          - *CustomLog*
            - *AppName*

3.3.3 Message File

Each event source contains information specific to the software that will be logging the events, such as a message file, as shown in table 2.

Sev	C	R	Facility	Code

Sev	Severity. The severity is defined as follows: 00 - Success 01 - Informational 10 - Warning 11 - Error
C	Customer bit. This bit is defined as follows: 0 - System code 1 - Customer code
R	Reserved bit.
Facility	Facility code. This value can be FACILITY_NULL
Code	Status code for the facility.

Table 3: Event Identifier [4]

3.3.4 Event Categories

Categories help organize events so that Event Viewer can filter them. Each event source can define its own numbered categories and the text strings to which they are mapped. Common event categories are: error, warning, information, success and failure.

3.3.5 Event Identifiers

Event identifiers uniquely identify a particular event. Each event source can define its own numbered events and the description strings to which they are mapped in its message file. Table 3 illustrates the format of an event identifier.

3.3.6 Event Log

Recorded Information about each event is stored in the event log, namely, event log record. The event log record includes time, type, and category information.

3.3.7 Event Data

Each event can have event-specific data associated with it. The Event Viewer does not interpret this data; it displays extra data only in a combined hexadecimal and text format. For example, many network-related events include network control blocks (NCB) as event-specific data.

3.4 Event Logging Model

The Event Logging Model is comprised of the components used for reading from and writing to the event logs. In the following, we illustrate how this is done and discuss the security related to reads and writes of the event logs.

3.4.1 Writing to the Event Log

When an application calls the *ReportEvent* function to write an entry to the event log, the system passes the parameters to the event-logging service. The event-logging service uses the information to write an *EVENTLOGRECORD* structure in the event log. Figure 1 illustrates this process.

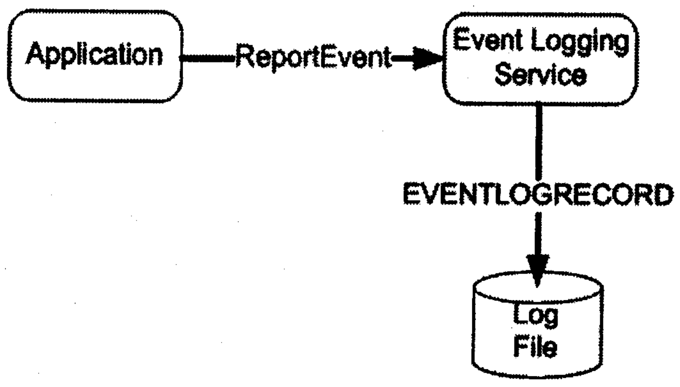

Figure 1: Writing to the Event Log

3.4.2 Reading from the Event Log

An event viewer application uses the *OpenEventLog* function to open the event log for an event source. The event viewer can then use the *ReadEventLog* function to read event records from the log. ReadEventLog returns a buffer containing an *EVENTLOGRECORD* structure and additional information that describes a logged event. Figure 2 illustrates this process.

3.4.3 Viewing the Event Log

When the user starts Event Viewer to view the event log entries, it calls the ReadEventLog function to obtain the EVENTLOGRECORD structures. The Event Viewer uses the event source and event identifier to get message text for each event from the registered message file (indicated by the EventMessageFile registry value for the source). The Event Viewer uses the LoadLibraryEx function

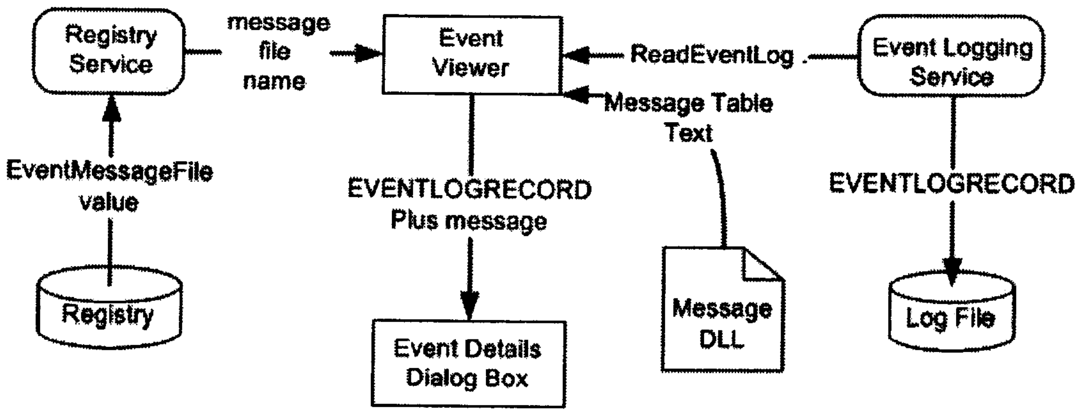

Figure 2: Reading From the Event Log

Access right	Description
ELF_LOGFILE_CLEAR (0x0004)	Required ClearEventLog
ELF_LOGFILE_READ (0x0001)	Required by OpenBackupEventLog and OpenEventLog
ELF_LOGFILE_WRITE (0x0002)	Required by RegisterEventSource

Table 4: Access Rights for Event Logging Functions [4]

to load the message file. The Event Viewer then uses the FormatMessage function to retrieve the base description string from the loaded module. Finally, the Event Viewer replaces the insertion parameters in the base description string to yield the final message string.

3.4.4 Event Logging Security

The Security log is designed for use by the system. However, users can read and clear the Security log if they have been granted the SE_SECURITY_NAME privilege which is the manage auditing and security log user right. As for write permission, only the Local Security Authority (Lsass.exe) has write permission for the Security log.

Access to the Application log, the System log, and custom logs is restricted. The system grants access based on the access rights granted to the account under which the thread is running. Table 4 shows which types of access are required by the event logging functions, and Table 5 describes the access rights granted for each account on each log for Windows XP/2000/NT.

Log	Account	Read	Write	Clear
Application	Administrators (system)	X	X	X
	Administrators (domain)	X	X	X
	LocalSystem	X	X	X
	Interactive user	X	X	
System	Administrators (system)	X	X	X
	Administrators (domain)	X		X
	LocalSystem	X	X	X
	Interactive user	X		
Custom	Administrators (system)	X	X	X
	Administrators (domain)	X	X	X
	LocalSystem	X	X	X
	Interactive user	X	X	

Table 5: Account Access Rights [4]

3.5 Types of Windows Logs

The log files that can be found on Windows are surprisingly many. However not all logs are enabled by default in order not to over-flood the storage space with logs. Following is a list of the logs that can be found on a Windows system as specified by [62].

3.5.1 Boot log

Windows has the option of creating a boot log, which regards numerous fine points of the startup process. This can be extremely valuable in various kinds of troubleshooting and system analysis.

To generate a boot log on a given occasion, bring up the Boot Menu during startup, select Option 2 (to create a boot log), then let Windows continue its startup. The file BOOTLOG.TXT is created in the root folder of the C: partition. If a boot log already exists, the old one is first backed up to a file named C:\ BOOTLOG.PRV (PRV meaning previous).

It is also possible to have a new boot log automatically generated every time Windows starts up by editing the C:\MSDOS.SYS file and adding the four lines shown below to MSDOS.SYS.

```
BootMenu=1
BootMenuDefault=2
BootMenuDelay=1
DisableLog=0
```

Line 1 guarantees that the Boot Menu comes up every time. Line 2 says the Boot Menu should always take option 2, which is to create a boot log. Line 4 may not be needed, but was recommended, and may be needed on some systems according to some peoples reports (or it may just be a precaution in case you had a DisableLog=1 that you missed earlier in the file). It prevents the disabling of the boot log.

Line 3 is the key one, and has advantages and disadvantages. Forcing the selection of option 2 will not work until the Boot Menu was up for some period of time. BootMenuDelay=1 says that the Boot Menu will appear and wait for 1 second, the minimum. But the disadvantage is that, if you want to manually bring up the Boot Menu for some other reason, you only have 1 second to pick the right number when the menu pops up, and that usually is not enough time. So you might want to set it a little higher, say, at 5 seconds, or whatever you want just knowing that this adds 5 seconds to your startup time.

All of the above, also, will slightly slow down startup. Again, we are talking only a few seconds. This is only true when a BOOTLOG.TXT or BOOTLOG.PRV already exists, and the slowdown is not in the forming of the boot log, but in the brief processes of deleting the existing .PRV file and backing up the existing BOOTLOG.TXT file before opening the new boot log.

3.5.2 Windows services

Task Scheduler service

The Task Scheduler service uses a log file, SchedLgU.txt, and the location of this file is specified in the LogPath registry value:

Key:HKLM\SOFTWARE\Microsoft\SchedulingAgent
Value: LogPath (REG_SZ)
Default value:%SystemRoot%\SchedLgU.txt (W2K, WXP),
%SystemRoot\Tasks\SchedLgU.txt (W2K3)

IPSEC Services/IPSEC Policy Agent service

The PolicyAgent service supports logging in a file named oakley.log, empty by default. To enable logging, the following registry value must be set to 1:

Key: HKLM \ SYSTEM \ CurrentControlSet \ Services \ PolicyAgent \ Oakley
Value: EnableLogging (REG_DWORD)

Additional logging can be specified by setting the following registry value to 1:

Key:HKLM\SYSTEM\CurrentControlSet\Services\PolicyAgent
Value: Debug (REG_DWORD)

The purpose of the IPSec Policy Agent is to retrieve policy information and pass it to other IPSec components that require this information to perform security services, as shown in figure 3.

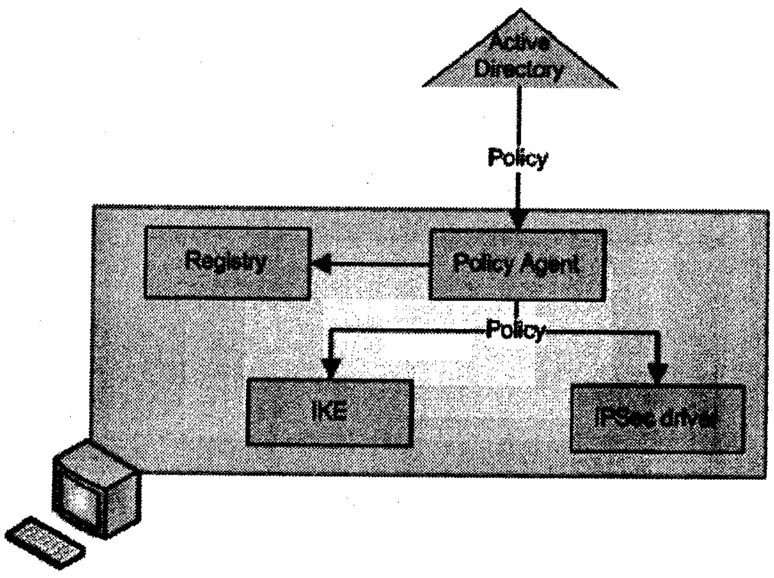

Figure 3: IPSEC Policy Agent Process

DNS Client service

The DNS Client service does not log by default. However, if a file named %systemroot%\system32\dnsrslvr.log is manually created, this file is used by the service to log debug information

DHCP Client service

Manages network configuration by registering and updating IP addresses and DNS names. A file named %systemroot%\system32\asyncreg.log can be manually created to enable logging of dnsapi functions.

Windows Time service

Windows Time service supports logging in a text file. The FileLogName registry value must be explicitly added:

Key: HKLM\SYSTEM\CurrentControlSet\Services\W32Time\Config
Value: FileLogName (REG_SZ)

Cluster service

Windows cluster service links individual servers so they can perform common tasks. Should any one server stop functioning, a process called failover automatically shifts its workload to another server to provide continuous service [63].

Windows Image Acquisition (WIA) service

Provides image acquisition services for scanners and cameras.

3.5.3 Windows setup

Following is a list of logs related to Windows setup:

- Windows installation log
- Windows installation errors log
- Information (.inf) files installation log
- COM+ setup log
- Windows domain configuration change log
- Log of administrative actions realized using the Configure Your Server Wizard
- Dynamic update log

3.5.4 Software updates

Software update that is managed by Windows, Windows Operating System, and security updates are all logged into the corresponding logs, listed below:

- Detailed list of software update managed by Windows Update
- Service Pack installation log
- Software update installation log
- Update Rollup Package installation log
- Software update slipstreaming log
- Software update log

- Catalog file registration log
- Windows XP pre-SP1 hotfixes log

3.5.5 Active Directory domain controllers

Domain Controller promotion (dcpromo.exe)

The dcpromo.exe program is used to promote or demote an Active Directory domain controller. When dcpromo is used, log files are generated.

Security Account Manager (SAM)

The sam.log file is used to log account lockout related events. When the SamLogLevel registry value is present and set to 1, the SAM creates a sam.log file:

> Key: HKLM\SYSTEM\CurrentControlSet\Control\Lsa
> Value: SamLogLevel (REG_DWORD)

Local Security Authority (LSA)

In Windows Server 2003, both the Kerberos authentication package and KDC service can be configured to log debug information, in a file named lsass.log. To enabled logging in a file, the LogToFile registry value must be set to 1:

> Key: HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Kerberos\Parameters
> Value: LogToFile (REG_DWORD)
> Content: 1 (to enable logging)

Then, the KerbDebugLevel registry value must be added and configured to specify what kind of Kerberos events must be logged:

> Key: HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Kerberos\Parameters
> Value: KerbDebugLevel (REG_DWORD)

Netlogon

The Netlogon service can be configured to log debugging information to a log file, named netlogon.log.

File Replication Service

The NT File Replication Service (NTFRS) service creates text-based log files in the %Systemroot%\Debug folder for troubleshooting NTFRS replication problems. The NTFRS service builds two types of text-based log files: Ntfrs_000n.log and Ntfrsapi.log. The Ntfrsapi.log file contains events that take place during promotion and demotion, namely creating the NTFRS registry keys. The Ntfrs_000n.log file stores transaction and event details in the Ntfrs_0001.log through Ntfrs_0005.log files. The most recent NTFRS transactions and events are written to the log with the highest version number in existence at that time [64].

3.5.6 Group Policy

The group policy log

- Group Policy Object Editor (Core specific entries)
- Group Policy Object Editor (CSE specific entries)
- Group Policy Folder Redirection CSE log
- Software Installation CSE log
- LSA API log used by GPO
- Security settings implemented during setup log

3.5.7 Internet Information Services (IIS)

During the installation of IIS 5.0, IIS 5.1 and IIS 6, events are logged in the iis5.log or iis6.log. By default, Internet services (HTTP, FTP, SMTP, NNTP) log requests in files stored under the LogFiles directory - Tracing for the RRAS service is typically enabled using netsh (set tracing command in the ras context). For each RRAS component that supports tracing, a registry key is stored under the Tracing key: `Key: HKLM\SOFTWARE\Microsoft\Tracing`
For each component, file logging is enabled when the `EnableFileTracing` registry value is set to 1 and when a tracing mask is specified in the `FileTracingMask` value.

3.5.8 WMI (Windows Management Instrumentation)

The WMI framework manages several log files. The Logging Directory registry value specifies the directory where these files are stored:

Key: HKLM\SOFTWARE\Microsoft\WBEM\CIMOM

Value: Logging Directory (REG_SZ) Default value: %SystemRoot%\system32\WBEM\Logs

3.5.9 Miscellaneous

- Log file for the SamChangePasswordUser2 API (used by the Change Password dialog box available after the Control-Alt-Delete sequence)
- User environment settings debugging System shutdown log
- Windows firewall log
- MS DTC service installation log
- Terminal Service installation log

3.6 Security Log

The security log contains the most important operating system events under a Windows operating system. Since it is the operating system log, all significant activities should be recorded there. given that the audit policy is configured properly. Thus we need to emphasize on the structure of this log and the meaning of the events logged. Documentation about the security log by Microsoft is not detailed enough to fully understand the events and their link between each other. The best reference we were able to find is an e-book by R.F Smith [3] which explains, to some extent, the meaning of the security events. To begin with, the security log is divided into nine categories based on the nine audit policies on a Windows system: Account Logon, Logon/Logoff, Account Management, Object Access, Detailed Tracking, System Events, Directory Service, Privilege Use, and Policy Change.

Account Logon

This category includes log-on attempts for local accounts stored in the local computer's SAM. On a Domain Controller (DC), this category will include all attempts to log on with a domain user account, regardless of where the log-on attempt originates from. Most common events you would see in this category is event ID 680 (Account used for logon by) and event ID 681 (Account logon failure). On a DC, you would expect to see events authentication and service tickets being granted, renewed, or revoked.

Logon/Logoff

Events logged for this category all attempted log-ons (success and failure) or log-offs to the local computer. If the computer is being used to log on to a DC, for example, that even will not be logged as part of this category. For the log-on failure events, the reason the log-on failed will be recorded, e.g. event ID 529 is logged whenever unknown username or bad passwords are known.

Account Management

When the Audit Account Management policy is enables, any changes related to user accounts and groups are logged. Example of such events are: password reset, newly created accounts, and users added or removed to/from a group.

Object Access

In Windows operating system, everything is classified as an object. For example, object access events would log access to files, folders, registry keys, printers, and services. By default most objects are not audited for the reason of not over flooding the logs. Auditing certain objects can easily be configured, however, only critical objects should be audited. The specific access to be audited is also configurable, depending on the sensitivity of the object. In most cases only write access is audited.

Detailed Tracking

Events logged in this category provides the ability to track programs executed on the system and to link those process events to log-on sessions logged by Logon/Logoff events and to file access events generated by the object access category. More details of the use of this category will be provided when discussing the log analysis.

System Events

Events directly related to the system are logged in this category. Such events would include Windows startup/shutdown, clearing of the audit log, and system time changes.

Directory Service

Directory Service events are related only to Active Directory and they provide low-level auditing for all types of objects in the Active Directory. Directory Service events not only identify the object that was accessed and by whom, but also document exactly which object properties were accessed, very similar to Object Access events. The two events that are logged in this category are event ID

565 (Object Open) and event ID 566 (Object Operation).

Privilege Use

Audits to user rights defined in the Local Security Policy are logged in this category. Three events are logged under this category: event ID 567 (Special privileges assigned to new logon), event ID 577 (Privilege service called), and event ID 578 (Privilege object operation).

Policy Change

The Policy Change category provides logged events of changes to important security policies, such as system's audit policy, or in the case of a Domain controller, to trust relationships.

Each event logged is a separate entry in the log file. The way an analyst or system administrator can link these events is by the process ID, logon ID, or handle ID. In the Windows event viewer, this can be done by searching through each event at a time. Thus a tool is needed to facilitate the analysis process. With the current ad-hoc based tools, an investigator must independently make decisions based oh his knowledge and experience.

It is possible to link the events using the unique identifiers that Windows generates for specific events. For example, when a user logs on a unique identifier (logon ID) is assigned for that logon session. Then any events triggered by that user are logged with the logon ID. If the user starts a process, the process is assigned a process ID. This event is logged with the user who created this process and the triggering process. For example, a user logs on and she is assigned logon ID = 123456. This user runs command.exe (Process ID = 8713) to run telnet (process ID = 991). If an analyst wanted to determine who/what ran telnet she would look at the telnet event which contains:

$$\begin{aligned}&ProcessID : 991\\&ImageFileName : telent.exe\\&CreatorProcessID : 8713\\&ClientLogonID : 123456\end{aligned}$$

Then the analyst would look for a logon event with logon ID = 123456 and find out who the user was and how did that user logon (logon type).

In this manner it is possible to link the events to get a detailed scenario of what has happened at a specific time or during a specific event. It should be noted that if the audit policy is not set properly some crucial events will be missing. For example, if an attacker modified a registry hive and set a malicious program to load at startup, the logs will not show any trace of this activity if there was no audit policy set on the related registry hive.

Chapter 4

Windows Events and Processes

In the previous chapter, the Windows logging system, log structure and the different logs available in a Windows environment were introduced. In this chapter we go further and discuss the Windows events and processes. In the first section we discuss how the logs can be modeled for use in our framework. Then we discuss some essential elements that will help in understanding the Windows processes. Examples of these elements are the basic security system components, as well as services and service accounts. In the following section, the concept of Log Functions is introduced. These functions are used in the log analysis model as well as in the implementation. Before proceeding further, it is essential to discuss first the different types of cases that a forensic investigator might face. The types of investigations and cases can vary tremendously from zero-knowledge cases to internal investigation cases. The zero-knowledge case is the worst case scenario. As its name implies, this refers to the case where the investigator has no information about the system specifics: types of applications, computer and audit policies, or networks. On the other hand, internal investigation refers to the case where the investigator knows specific details about the environment she is about to analyze. The latter case relates more to corporate fraud and legal cases against a company employee.

When dealing with a zero-knowledge case, the investigator has to assume an uncontrolled environment where audit policies are probably not set and logging configurations are the default configuration as set by the software or operating system upon installation. Furthermore, depending on the suspect's computer skills, the logs can be tampered with or cleared. Thus the investigator needs to establish the authenticity and congruency of the logged events. Certain log properties can be used in this case such as the sequence of events, time gaps, meaningful correlation between logs if possible, and existing events that are expected to be in a log.

In such a case, the investigator has to explore the logs and collect data that may lead to the evidence required to prove or disprove accusations against the suspect. Techniques which the investigator can use to carry out this task may vary. The end result is what matters the most. The investigator has to gather sufficient evidence to back up his conclusion and formalize the findings to be represented in a court of law. In this thesis we propose a method to deal both with gathering the evidence and formalizing the facts.

In a corporate or internal investigation, the investigating party can be either internal to the company or an external third party. In either case, the investigator is given insight about the system at hand by the IT or system administrator. This information can be extremely important in aiding the investigation: the type of operating system, patches, network, audit and logging policy, and critical resources. By knowing specifics about the system at hand, the investigator knows what to look for and what to expect. For example, a security or computer policy can be modeled and compared with the logged events. Any behavior out of the norm will be located instantaneously.

In 1997, Marcus Ranum developed a theory called Artificial Ignorance (AI), which states that if you remove all legitimate activity from the Web server logs, what you have left should be unusual [5]. Such a technique is a common practice for anomaly detection mainly for IDS logs. Bringing this theory into practice for forensic analysis can reveal most or part of the malicious behavior. Given the policy and architecture of the system, an investigator can build a model of that system in normal working conditions. The seized logs can be also modeled and a comparison of the two models is done. The difference of the two models should be the abnormal activity the investigator is looking for.

Through our proposed approach we present different methods to deal with the different types of investigations. We also present the methodology to construct models which can be used for baselining or to apply the Artificial Ignorance theory or to construct "attack signatures". Furthermore, we present, through our approach, event correlation between different log sources. On any system, numerous logs can be obtained and can relate to each other in some way. Upon the correlation of log files, a new trace of events can be obtained revealing information about the period at which a malicious action has taken place. Correlation can be done in two ways: correlation based on time only, or correlation based on time and specific properties. In the latter case, a property is defined as the logical sequence of events as they are supposed to happen in normal circumstances. For example, a new process creation event in the security log should be followed by a service start event in the system log followed by a service started event in the application log, given that the new process created is a service. The time gap between these events, ideally, should be less then one minute. For specific properties and time based correlation a comprehensive knowledge of the system and sequence of events is required. However, when dealing with terabytes of log files, having these properties will facilitate the work tremendously.

4.1 Modeling Logs

In this section we show how logs can be modeled. To begin with, we model the normal behavior of a system. A normal system is a system that has not been tampered with and contains no malicious activity of any kind. In other words, the system behaves in a contingent and safe manner. Our model is based on a Windows XP - service pack 2 operating system. It should be noted that there might be minor changes between different versions of Windows operating system, but the main concepts remain the same. We assume that the audit policy is set to audit all nine categories of the security log (account logon, account management, detailed tracking, directory services, logon/logoff, object access, policy change, privilege use, and system events) for both failure and successful events. Furthermore, the Windows Firewall is configured and logging is enabled. The model we present is based on the correlation of four logs: security log, system log, application log, and the firewall log. Although more logs can be correlated, we simplify our model by limiting the correlation to only four logs as a proof of concept. As will be shown in the model, the security log contains the most pertinent events and is used as a backbone for our correlation. Every event that occurs on the system has at least one corresponding event logged in the security log. This fact is useful to determine any log tampering. Many attackers try to cover their tracks by deleting the logs. However, due to the abundant logs on a Windows machine, the attacker might not be able to delete all logs. Moreover, if proper log management is implemented and maintained, the attacker will not be able to destroy all the logs.

The model consists of a log trace, which can be numerous traces combined in the proper sequence. A trace represents a run of the system, which is a sequence of consecutive events. A complete run is defined as the run beginning when the Windows operating system loads till it shuts down. We divide a complete run into smaller multiple runs, thus providing an easier modeling technique. The small runs or traces are logically combined to model a complete run, or a segmented run of interest to the examiner. Examples of a complete run can be startup, service initialization, logon initialization, user logon, firewall initialization, processes executing, network connections, object use, and system shutdown. However, in order to properly model the exact system behavior, a comprehensive knowledge of processes and event sequences of the specific operating system at hand is required.

In the following we present the background information needed to build the model and show how events are modeled and correlated.

Security System Components

In order to fully grasp Windows processes, we first need to discuss the security system components that are responsible of Window's low level security. The core components and databases that implement Windows security, are [2]:

- **Security reference monitor (SRM)** - (\Windows\System32\Ntoskrnl.exe) responsible for defining the access token data structure, performing security access checks on objects, manipulating privileges/user rights, and generating corresponding security audit messages.
- **Local security authority subsystem (Lsass)** - (\Windows\System32\Lsass.exe) a user-mode process that is responsible for the local system security policy, user authentication, and sending security audit messages to the Event Log. Lsass loads the local security authority service (Lsasrv-\Windows\System32\Lsasrv.dll), which implements most of this functionality. Lsass uses a policy database (stored in the registry under HKLM\SECURITY), which is a database that contains the local system security policy settings. It contains information such as trusted domains used to authenticate logon attempts, permissions for system access and the permitted method (interactive, network, and service logons), privileges, and the security auditing to be performed.
- **Security Accounts Manager (SAM) service** - the SAM is a set of subroutines responsible for managing the database containing the usernames and groups defined on the local machine. It is implemented as \Windows\System32\Samsrv.dll which runs in the Lsass process. The SAM database is stored in the registry under HKLM\SAM. The database that contains the defined local users and groups, along with their passwords and other attributes. However, on domain controllers the SAM stores the system's administrator recovery account definition and password.
- **Active Directory** (\Windows\System32\Ntdsa.dll, runs in the Lsass process) - a directory service containing a database that stores information about objects in a domain. The domain is defined as a collection of computers and their associated security groups. They are managed as a single entity. The objects in a domain include users, groups, and computers. The Active Directory is replicated across the computers that are designated as domain controllers of the

domain.

- **Authentication packages** - dynamic-link libraries (DLLs) that run both in the context of the Lsass process and client processes and that implement Windows authentication policy. An authentication DLL checks whether a given username and password match. If so, it returns information detailing the user's security identity to the Lsass. LSASS then uses that information to generate a token.

- **Logon process** (Winlogon - \Windows\System32\Winlogon.exe) - a user-mode process that is responsible for responding to the SAS and for managing interactive logon sessions.

- **Graphical Identification and Authentication (GINA)** - A user-mode DLL that runs in the Winlogon process and that Winlogon uses to obtain a user's name and password or smartcard PIN. The standard GINA is \ Windows\System32\Msgina.dll.

- **Network logon service** (Netlogon - \Windows\System32\Netlogon.dll) - a service that sets up the secure channel to a domain controller to send security requests, such an interactive logon or LAN Manager and NT LAN Manager (v1 and v2) authentication validation.

- **Kernel Security Device Driver** (KSecDD - \Windows\System32\Drivers\Ksecdd.sys) - a kernel-mode library that implements the local procedure call (LPC) interfaces. It is used by kernel-mode security components to communicate with Lsass in user mode.

Logon Process

The logon process is composed of multiple steps and involves multiple processes which are triggered during a logon process. In order to understand the exact process we have to understand tokens, impersonation, account rights and privileges [2]. We discuss these components prior to discussing the actual logon process. Then we explain the process involved during a *winlogon* and a *userlogon*.

Tokens

To identify the security context of a process or thread the Security Reference Monitor (SRM) uses a token, also known as access token. A security context contains the privileges, accounts, and groups associated with the process or thread. When a logon occurs, Winlogon creates an initial token to represent the user logging on and attaches the token to the initial process it starts, by default, Userinit.exe. Every processes created afterwards, by default, inherits a copy of the token of its creator, therefore all processes in the user's session run under the same token.

Two components of the token are used by the security mechanisms in Windows to determine object access and secure operations. The first component is the token's user account SID and group SID fields and the second component is the privilege array.

Impersonation

Impersonation is used mainly in a client/server programming model. This feature enables a server to temporarily adopt the security profile of a client making a resource request. The server can then access resources on behalf of the client, and the SRM carries out the access validations. Usually, a server has access to more resources than a client does and loses some of its security credentials during impersonation. However, the reverse can be true: the server can gain security credentials during impersonation.

After the server thread finishes its task, it reverts to its primary security profile. These forms of impersonation are convenient for carrying out specific actions at the request of a client and for ensuring that object accesses are audited correctly. However, impersonation does not hold for the entire run of an application. If an entire application must execute in a client's security context or must access network resources, the client must be logged on to the system. The LogonUser Windows API function enables this action. A server thread can adopt the token as an impersonation token, or the server can start a program that has the client's credentials as its primary token. From a security standpoint, a process created using the token returned from an interactive logon via LogonUser looks like a program a user starts by logging on to the machine interactively.

Account Rights

Account Rights are not enforced by the SRM, nor are they stored in tokens. The function responsible for logon is LsaLogonUser. WinLogon, for example, calls the LogonUser API when a user logs on interactively to a computer and LogonUser calls LsaLogonUser. The function takes a parameter that indicates the type of logon being performed, which includes interactive, network, batch, service, terminal server client, and unlock.

In response to logon requests, the LSA retrieves account rights assigned to a user from the LSA policy database at the time that a user attempts to log on to the system. LSA checks the logon type against the account rights assigned to the user account logging on and denies the logon if the account does not have the right that permits the logon type or it has the right that denies the logon type. Table 6 lists the user rights defined by Windows.

User Right	Role
Deny logon interactively, Allow logon interactively	Used for interactive logons that originate on the local machine
Deny logon over the network, Allow logon over the network	Used for logons that originate from a remote machine
Deny logon through Terminal Services, Allow logon through Terminal Service	Used for logons through a Terminal Server client
Deny logon as a service, Allow logon as a service	Used by the Service Control Manager when starting a service in a particular user account
Deny logon as a batch job, Allow logon as a batch job	Used when performing a logon of type batch

Table 6: Account Rights [5]

Privileges

The number of privileges defined by the operating system has grown over time. Unlike user rights, which are enforced in one place by LSA, different privileges are defined by different components and enforced by those components. For example, the debug privilege, which allows a process to bypass security checks when opening a handle to another process with the OpenProcess Windows API, is checked for by the Process Manager. Table 7 and 8 is a full list of privileges, and it describes how and when system components check for them.

When a component wants to check a token to see whether a privilege is present, it uses the PrivilegeCheck or LsaEnumerateAccountRights APIs if running in user mode and SeSinglePrivilegeCheck or SePrivilegeCheck if running in kernel mode. The privilege-related APIs are not account-right aware, but the account-right APIs are privilege-aware.

Unlike account rights, privileges can be enabled and disabled. For a privilege check to succeed, the privilege must be in the specified token and it must be enabled. The idea behind this scheme is that privileges should be enabled only when their use is required so that a process cannot inadvertently perform a privileged security operation.

Logon

Interactive logon occurs through the interaction of: (1) the logon process Winlogon, (2) Lsass, (3) one or more authentication packages, and (4) the SAM or Active Directory.

Authentication packages are DLLs that perform authentication checks. The two main authentication packages are Kerberos and MSV1_0. Kerberos is responsible for interactive logon to a domain, and MSV1_0 is responsible for interactive logon to a local computer, for domain logons to trusted pre-Windows 2000 domains, and for when no domain controller is accessible.

Winlogon is a trusted process that coordinates logon, starts the user's first process at logon,

Privilege	User Right	Privilege Use
SeAssignPrimaryTokenPrivilege	Replace a process-level token	Checked for by various components, such as NtSetInformationJob, that set a process's token.
SeAuditPrivilege	Generate security audits	Required to generate events for the Security event log.
SeBackupPrivilege	Backup files and directories	Causes NTFS to grant the following access to any file or directory, regardless of the security descriptor that's present: READ_CONTROL, ACCESS_SYSTEM_SECURITY, FILE_GENERIC_READFILE_TRAVERSENote that when opening a file for backup, the caller must specify the FILE_FLAG_BACKUP_SEMANTICS flag.
SeChangeNotifyPrivilege	Bypass traverse checking	Used by NTFS to avoid checking permissions on intermediate directories of a multilevel directory lookup.
SeCreateGlobalPrivilege	Create global objects	Required for a process to create section & symbolic link objects in the directories of the Object Manager namespace are assigned to different session than the caller
SeCreatePagefilePrivilege	Create a pagefile	Checked for by NtCreatePagingFile, which is the function used to create a new paging file.
SeCreatePermanentPrivilege	Create permanent shared objects	Checked for by the object manager when creating a permanent object.
SeCreateTokenPrivilege	Create a token object	NtCreateToken, the function that creates a token object, checks for this privilege.
SeDebugPrivilege	Debug programs	If the caller has this privilege enabled, Process Manager allows access to any process using NtOpenProcess, regardless of the process's security descriptor.
SeImpersonatePrivilege	Impersonate a client after authentication	Process Manager checks for this when a thread wants to use a token for impersonation and the token represents a different user than the thread's process token.
SeIncreaseBasePriorityPrivilege	Increase scheduling priority	Checked for by the Process Manager. Required for raising process priority
SeIncreaseQuotaPrivilege	Adjust memory for a process	Enforced when changing a process's working set thresholds, and a process's paged and nonpaged pool quotas.
SeLoadDriverPrivilege	Load and unload device drivers	Checked for by the NtLoadDriver and NtUnload- Driver driver functions.
SeLockMemoryPrivilege	Lock pages in memory	The kernel implementation of VirtualLock.

Table 7: Windows Account Privileges 1/2 [3]

Privilege	User Right	Privilege Use
SeMachineAccountPrivilege	Add workstations to the domain	Checked for by the Security Accounts Manager on a domain controller when creating a machine account in a domain
SeManageVolumePrivilege	Perform volume maintenance tasks	Enforced by file system drivers during a volume open operation, required to perform disk checking and defragmenting activities.
SeProfileSingleProcessPrivilege	Profile single process	Checked for by prefetcher's function that returns prefetch information for an individual process.
SeRemoteShutdownPrivilege	Force shutdown from a remote system	Winlogon checks that remote callers of the InitiateSystemShutdown function have this privilege.
SeRestorePrivilege	Restore files & directories	Causes NTFS to grant the following access to any file or directory regardless of the security descriptor: WRITE_DAC, WRITE_OWNER, ACCESS_SYSTEM_SECURITY, FILE_GENERIC_WRITE, FILE_ADD_FILE, FILE_ADD_SUBDIRECTORY, DELETENote
SeSecurityPrivilege	Manage auditing & security log	Required to access the SACL of a security descriptor, read & clear the security event log.
SeShutdownPrivilege	Shut down the system	This privilege is checked for by NtShutdownSystem and NtRaiseHardError, which presents a system error dialog box on the interactive console.
SeSyncAgentPrivilege	Synchronize directory service data	Required for the LDAP directory synchronization services and allows the holder to read all objects and properties in the directory.
SeSystemEnvironmentPrivilege	Modify firmware environment variables	Required by NtSetSystemEnvironmentValue and NtQuerySystemEnvironmentValue to modify and read firmware environment variables using the HAL.
SeSystemProfilePrivilege	Profile system performance	Checked for by NtCreateProfile, the function used to perform profiling of the system.
SeSystemtimePrivilege	Change system time	Required to change the time or date.
SeTakeOwnership	Take ownership of object	Required to take ownership of an object without being granted discretionary access
SeTcbPrivilege	Act as part of the operating system	Checked for by the Security Reference Monitor when the session ID is set in a token, by the Plug and Play Manager for Plug and Play event creation and management, the Windows 2000 implementation of LogonUser, BroadcastSystemMessageEx when called with BSM_ALLDESKTOPS, and LsaRegisterLogonProcess.
SeUndockPrivilege	Remove computer from a docking station	Checked for by the user-mode Plug and Play Manager when either a computer undock is initiated or a device eject request is made.

Table 8: Windows Account Privileges 2/2 [3]

handles logoff, and manages various other operations relevant to security. The Winlogon process must ensure that operations relevant to security are not visible to any other active processes.

Graphical Identification and Authentication (GINA) DLL is a dll used by Winlogon to obtain a user's account name and password. The default GINA is Msgina (\Windows\System32\ Msgina.dll). Msgina presents the standard Windows logon dialog box. After obtaining a username and password from the GINA, Winlogon calls Lsass to authenticate the user attempting to log on. If the user is authenticated, the logon process activates a logon shell on behalf of that user. The interaction between the components involved in logon is illustrated in Figure 4.

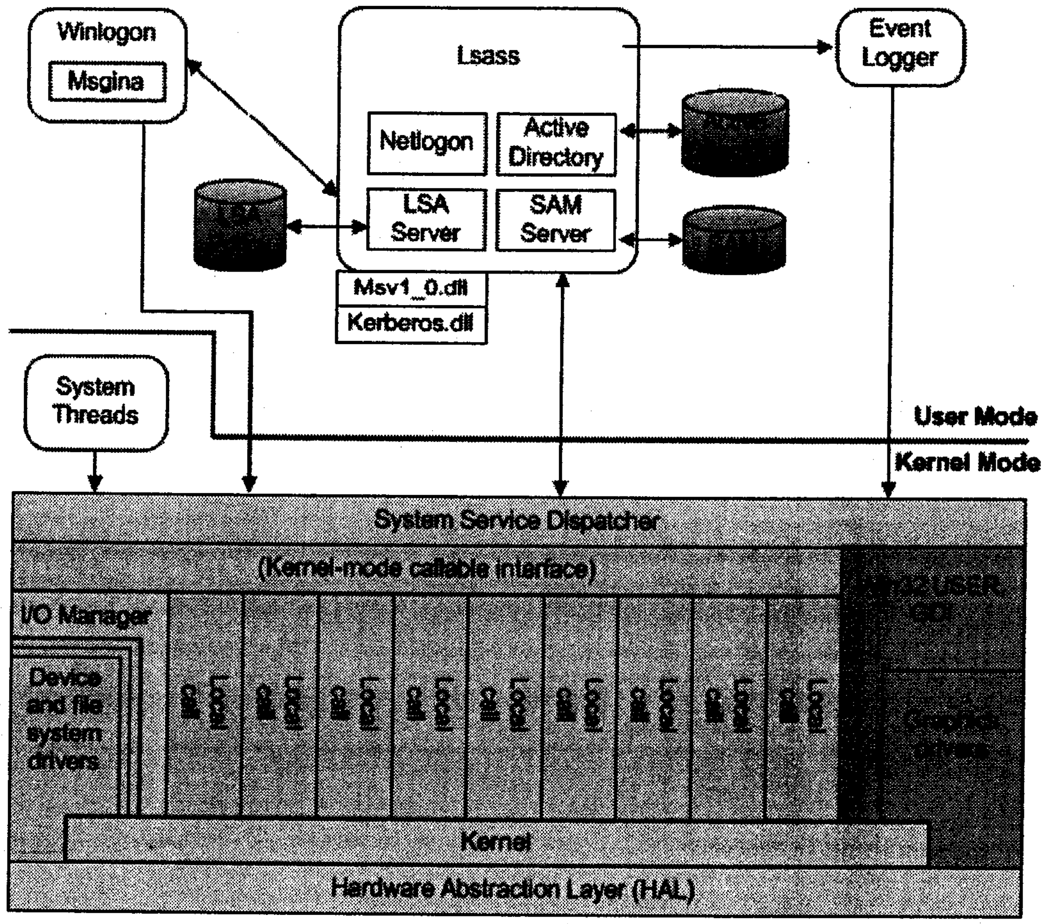

Figure 4: Interaction between Windows Logon Components [2]

Winlogon Initialization

During system initialization, i.e. before any user applications are active, Winlogon performs the following steps to ensure that it controls the workstation once the system is ready for user interaction

[2]:

1. Creates and opens an interactive window station to represent the keyboard, mouse, and monitor. Winlogon creates a security descriptor for the station that has one and only one ACE containing only the System SID, which ensures that no other process can access the workstation unless explicitly allowed by Winlogon.

2. Creates and opens two desktops: an application desktop and a Winlogon desktop. The security on the Winlogon desktop is created so that only Winlogon can access that desktop. The other desktop allows both Winlogon and users to access them. By doing so any time the Winlogon desktop is active, no other process has access to any active code or data associated with the desktop. Windows uses this feature to protect the secure operations that involve passwords and locking and unlocking the desktop.

3. Before anyone logs on to a computer, the visible desktop is Winlogon's. After a user logs on, pressing Ctrl+Alt+Delete switches the desktop from Default to Winlogon. Thus, the SAS always brings up a secure desktop controlled by Winlogon.

4. Establishes an LPC connection with Lsass's LsaAuthenticationPort. This connection will be used for exchanging information during logon, logoff, and password operations and is made by calling LsaRegisterLogonProcess.

   Winlogon then performs the following Windows operations to set up the window environment:

5. Initializes and registers a window class data structure that associates a Winlogon procedure with the window it subsequently creates.

6. Registers the SAS associating it with the window just created, guaranteeing that Winlogon's window procedure is called whenever the user enters the SAS. This measure prevents Trojan horse programs from gaining control of the screen when the SAS is entered.

7. Registers the window so that the procedure associated with this window gets called if a user logs off or if the screen saver times out. The Windows subsystem checks to verify that the process requesting notification is the Winlogon process.

Once the Winlogon desktop is created during initialization, it becomes the active desktop. When the Winlogon desktop is active, it is always locked. Winlogon unlocks its desktop only to switch to the application desktop or the screen-saver desktop. (Only the Winlogon process can lock or unlock a desktop.)

User Logon Steps

When a user presses the SAS (Ctrl+Alt+Delete) the logon process begins in the following steps, assuming that the correct credentials are provided at login [2]:

- Winlogon:
  - Calls the GINA to obtain a username and password and
  - Creates a unique local logon SID for this user that it assigns to this instance of the desktop.
  - Passes this SID to Lsass as part of the LsaLogonUser call, this SID will be included in the logon process token.
  - Retrieves a handle to a package by calling the Lsass function LsaLookupAuthentication-Package. Authentication packages are listed in the Registry under HKLM\SYSTEM\CurrentControlSet\Control\Lsa.
  - Passes logon information to the authentication package via LsaLogonUser. As discussed earlier, two authentication packages are mainly used: MSV1_0 and Kerberos.

    Case MSV1_0:

    The MSV1_0 authentication package takes the username and a hashed version of the password and sends a request to the local SAM to retrieve the account information, which includes the password, the groups to which the user belongs, and any account restrictions. MSV1_0 first checks the account restrictions, such as hours or type of accesses allowed. If the user can't log on because of the restrictions in the SAM database, the logon call fails and MSV1_0 returns a failure status to the LSA.

    MSV1_0 then compares the hashed password and username to that stored by the SAM. In the case of a cached domain logon, MSV1_0 accesses the cached information by using Lsass functions that store and retrieve "secrets" from the LSA database (the SECURITY hive of the registry). If the information matches, MSV1_0 generates an LUID for the logon session and creates the logon session by calling Lsass, associating this unique identifier with the session and passing the information needed to ultimately create an access token for the user.

    If MSV1_0 needs to authenticate using a remote system, as when a user logs on to a trusted pre-Windows 2000 domain, MSV1_0 uses the Netlogon service to communicate with an instance of Netlogon on the remote system. Netlogon on the remote system interacts with the MSV1_0 authentication package on that system, passing back authentication results to the system on which the logon is being performed.

Case Kerberos:

The basic control flow for Kerberos authentication is the same as the flow for MSV1_0. However, in most cases, domain logons are performed from member workstations or servers (rather than on a domain controller), so the authentication package must communicate across the network as part of the authentication process. The package does so by communicating via the Kerberos TCP/IP port (port 88) with the Kerberos service on a domain controller. The Kerberos Key Distribution Center service (Windows\System32\ Kdcsvc.dll), which implements the Kerberos authentication protocol, runs in the LSASS process on domain controllers. After validating hashed username and password information with Active Directory's user account objects (using the Active Directory server Windows\System32\Ntdsa.dll), Kdcsvc returns domain credentials to LSASS, which returns the result of the authentication and the user's domain logon credentials (if the logon was successful) across the network to the system where the logon is taking place.

- After a logon has been authenticated, LSASS looks in the local policy database for the user's allowed access, including interactive, network, batch, or service process. If the requested logon doesn't match the allowed access, the logon attempt will be terminated. LSASS deletes the newly created logon session by cleaning up any of its data structures and then returns failure to Winlogon, which in turn displays an appropriate message to the user. If the requested access is allowed, LSASS adds the appropriate additional security IDs (such as Everyone, Interactive, and the like). It then checks its policy database for any granted privileges for all the IDs for this user and adds these privileges to the user's access token.
- LSASS then calls the executive to create the access token.
- The executive creates a primary access token for an interactive or service logon and an impersonation token for a network logon.
- After the access token is successfully created, LSASS duplicates the token, creating a handle that can be passed to Winlogon, and closes its own handle. If necessary, the logon operation is audited. At this point, LSASS returns success to Winlogon along with a handle to the

access token, the LUID for the logon session, and the profile information, if any, that the authentication package returned.

- Winlogon then looks in the registry at the value HKLM\SOFTWARE\Microsoft\Windows NT\Current Version\Winlogon\Userinit and creates a process to run whatever the value of that string is. The default value is Userinit.exe, which loads the user profile and then creates a process to run whatever the value is of HKCU\SOFTWARE\Microsoft\Windows NT\Current Version\Winlogon\Shell, if that value exists. That value does not exit by default. If it doesn't exist, Userinit.exe does the same for HKLM\SOFTWARE\Microsoft\Windows NT\Current Version\Winlogon\Shell, which defaults to Explorer.exe. Userinit then exits (which is why Explorer.exe shows up as having no parent when examined in Process Explorer.

Services and Service Accounts[65]

Services are a very attractive door for attacks since services run unattended at startup, a user can be unaware that a service is running. For example, Internet worms such as Nimda exploit the fact that users can unknowingly run Web servers on their workstations. The infected workstations spread the worm to thousands of computers across the Internet.

Furthermore, security exposure occurs whenever a service is configured to log on as a user. The user name and password information for each service that uses a domain or local user account is stored in the registry. Any user having administrative access to the computer can easily exploit this vulnerability.

System Accounts

A service must log on as an account to access resources and objects on the operating system. If you assign an account to a service that does not have appropriate permissions to log on, the Services snap-in for the Microsoft Management Console (MMC) automatically grants that account the required Log on as a Service user right on the computer being managed. Microsoft Windows Server 2003 includes the following three built-in local accounts used as the logon accounts for various system services:

- Local System account

  The Local System account is a powerful account that has full access to the computer, including the directory service when used for services running on domain controllers. It can start a service and provide the security context for that service. It The account acts as the host computer account on the network and as such has access to network resources just like any other domain account. The actual name of the account is NT AUTHORITY\System, on the network, this account appears as DOMAIN\ < *machine name* >. If a service logs on using the Local System account on a domain controller, it has Local System access on the domain controller itself, which, if compromised, could allow malicious users to change anything in the domain they wanted.

- Local Service account

  The Local Service account has reduced privileges similar to an authenticated local user account. This limited access helps safeguard the computer if an attacker compromises individual services or processes. The actual name of the account is NT AUTHORITY\LocalService.

- Network Service account

  The Network Service account has reduced privileges similar to an authenticated user account. This limited access helps safeguard the computer if an attacker compromises individual services or processes. A service that runs as the Network Service account accesses network resources using the credentials of the computer account in the same manner as a Local System service does. The actual name of the account is NT AUTHORITY\NetworkService.

User Accounts

Several categories of user accounts can log on as a service. Each category has its own capabilities and privileges:

- Local User Accounts

  This category contains the accounts that you create locally on a computer. These accounts have very limited privileges on the local computer unless you specifically grant them higher privileges or add them to groups that already possess those privileges.

- Local Administrator Accounts

  This category includes the built-in Administrator account that you create and use when you first install Windows. It includes any other user accounts that you subsequently create and add to the built-in Administrators group. Members of this group have full and unrestricted access to the local computer.

- Domain User Accounts

  This category includes accounts that you create in the domain, for example, by using the

Service Name	Log On As
Alerter	Local Service
Application Layer Gateway Service	Local Service
Remote Registry	Local Service
Smart Card	Local Service
TCP/IP NetBIOS Helper	Local Service
Telnet	Local Service
Uninterruptible Power Supply	Local Service
WebClient	Local Service
Windows Image Acquisition (WIA)	Local Service
Windows Time	Local Service
WinHTTP Web Proxy Auto-Discovery Service	Local Service
DHCP Client	Network Service
Distributed Transaction Coordinator	Network Service
DNS Client	Network Service
License Logging	Network Service
Performance Logs and Alerts	Network Service
Remote Procedure Call (RPC) Locator	Network Service

Table 9: Service Accounts Settings [2]

Active Directory Users and Computers management console. These accounts have limited privileges in the domain unless you specifically grant them higher privileges or add them to groups that already possess those privileges.

- Domain Administrator Account

  This category includes the built-in domain Administrator account that you create and use when you first install Active Directory. It includes any other user accounts that is subsequently created and added to the built-in local Administrators group or to the Domain Admins or Enterprise Admins groups. Members of these groups have complete and unrestricted access to the domain and, in the case of the Enterprise Admins group, to the entire forest.

Instead of using the Local System account, many common services now use the Local Service or Network Service account. These accounts have a much lower level of privileges than the Local System account and, therefore, present a lower security threat. Some services and their default logon accounts are listed in table 9.

4.2 Log Functions

In this section we present how event functions can be used to represent events in a more readable and usable form. Having a function called $SuccessfulLogon$, for example, is easier to understand and use over calling it $EventID528$. Furthermore, these functions are that same functions used in the implementation. In the following we present some event functions from the security, application, system and firewall logs. Following each event function is an explanation to illustrate its use.

4.2.1 Security Log

Domain Controller Functions

The following functions will only be logged on domain controllers through the use of Kerberos authentication system. In Windows categorization, these are under the category of Account Logon.

$$AuthenticationTicketGranted(user, realmName, userID, serviceName, serviceID,$$
$$ticketOptions, preAuthenticationType, clientAddress, sORf) \ - \ EventID\ 672$$

This event only appears on domain controllers when a Kerberos TGT is granted and is shown typically when a server restarts or when a workstation boots up. Three fields are of main importance, *user*, *realmName*, and *clientAddress*. The *user* along with the *realmName* are used to identify the user who logged on. The *userID* is the user name and the realm name in the NT format, for example, "S-1-5-21-2121316058-685099279-904526279-500". The *clientAddress* is the IP address from where the user logged on. Finally the *sORf* field represents a success or failure audit. This field is common for all the functions/events.

$$ServiceTicketGranted(user, domain, serviceName, ticketOptions, clientAddress, sORf)$$
$$- \ EventID\ 673$$

A service ticket is obtained any time a user or computer accesses a server on the network. The *user* and *domain* identify the user or computer accessing the server, and the *clientAddress* specifies the IP address of the user. The *serviceName* identifies the server name.

$$TicketGrantRenewed(user, domain, serviceName, ticketOptions, clientAddress, sORf)$$
$$- \ EventID\ 674$$

This event is logged whenever the ticket expires, as Kerberos limits the time a ticket can be valid. The fields are the same as ServiceTicketGranted.

$$PreAuthenticationFailed(user, serviceName, preAuthenticationType, failureCode,$$
$$clientAddress) \ - \ EventID\ 675$$

When a user attempts to log on to a domain controller and fails for some reason, this event will be recorded on the domain controller's logs. The reason for this log-on failure is recoded in the $failureCode$ field. The failure codes are based on RFC 1510, Kerberos Network Authentication Service (V5), and are shown in table 10.

In some cases the user's initial logon might fail due to reasons other then those mentioned in table 10. When this happens AuthenticationTicketRequestFailed is logged.

$$AuthenticationTicketRequestFailed(user, realmName, serviceName, failureCode,$$
$$clientAddress) \; - \; EventID\, 676$$
$$ServiceTicketRequestFailed(user, realmName, serviceName, failureCode,$$
$$clientAddress) \; - \; EventID\, 677$$

$$LogonAttempt(user, logonAccount, logonAttemptBy, sORf) \; - \; EventID\, 680$$

A LogonAttempt event is used for NTLM authentication attempts and is logged on both the domain controller and the local machine. This event is used frequently when a service needs to execute under certain privileges. In such as case, this event is used to log on the account with the privileges authorized for the service, and the service is given a token with the corresponding privileges. The logonAttemptBy field identifies the authentication package that processed the authentication request.

Logon/Logoff

$$SuccessfulLogon(user, session, logonProcess, typeOfLogon) \; - \; EventID\, 528$$

Whenever an account logs on to a local computer using a local SAM account or a domain logon, this events appears in the security log. The *user* indicates the account used for logon. In a given log a user may have several logon sessions at different times. Each session is identified by a logon session identifier, *session*. The *logonProcess* can be any of: KSecDD (Kernel Security Device Driver), RASMAN (Remote Access Service Manager), Secondary Logon Service, LAN Manager Workstation Service, CHAP, DCOMSCM, Winlogon, or Winlogon\MSGina. As for the $typeOfLogon$ field, this specifies which type of logon was used such as interactive or remote. A complete list of logon types are shown in table 11.

Failure Code	Kerberos RFC Description	Notes
1	Client's entry in database has expired	
2	Server's entry in database has expired	
3	Requested protocol ver ♯ not supported	
4	Client's key encrypted in old master key	
5	Server's key encrypted in old master key	
6	Client not found in Kerberos database	Bad user name, or new computer/user account has not replicated to DC yet
7	Server not found in Kerberos database	New computer account has not replicated yet or computer is pre-w2k
8	Multiple principal entries in database	
9	The client or server has a null key	Administrator should reset the password on the account
10	Ticket not eligible for postdating	
11	Requested start time is later than end time	
12	KDC policy rejects request	Workstation/logon time restriction
13	KDC cannot accommodate requested option	
14	KDC has no support for encryption type	
15	KDC has no support for checksum type	
16	KDC has no support for padata type	
17	KDC has no support for transited type	
18	Clients credentials have been revoked	Account disabled, expired, or locked out
19	Credentials for server have been revoked	
20	TGT has been revoked	
21	Client not yet valid - try again later	
22	Server not yet valid - try again later	
23	Password has expired	The users password has expired
24	Pre-authentication information is invalid	Usually means bad password
25	Additional pre-authentication required	
31	Integrity check on decrypted field failed	
32	Ticket expired	Frequently logged by computer accounts
33	Ticket not yet valid	
34	Request is a replay	
35	The ticket isn't for us	
36	Ticket and authenticator don't match	
37	Clock skew too great	Workstations clock too far out of sync with the DCs
38	Incorrect net address	IP address change
39	Protocol version mismatch	
40	Invalid msg type	
41	Message stream modified	
42	Message out of order	
44	Specified version of key is not available	
45	Service key not available	
46	Mutual authentication failed	May be a memory allocation failure
47	Incorrect message direction	
48	Alternative authentication method required	
49	Incorrect sequence number in message	
50	Inappropriate type of checksum in message	
60	Generic error (description in e-text)	
61	Field is too long for this implementation	

Table 10: Kerberos Network Authentication Service Failure Codes [2]

Logon Type	Description
2	Interactive (logon at keyboard and screen of system)
3	Network (i.e. connection to shared folder on this computer from elsewhere on network or IIS logon - Never logged by 528 on W2k and forward. See event 540)
4	Batch (i.e. scheduled task)
5	Service (Service startup)
7	Unlock (i.e. unattended workstation with password protected screen saver)
8	NetworkCleartext (Logon with credentials sent in the clear text. Most often indicates a logon to IIS with "basic authentication")
9	NewCredentials
10	RemoteInteractive (Terminal Services, Remote Desktop or Remote Assistance)
11	CachedInteractive (logon with cached domain credentials such as when logging on to a laptop when away from the network)

Table 11: Logon Types [3]

$$LogonFailure(user, typeOfLogon, logonProcess, reason)$$

There are multiple events logged that indicate an account logon failure. The EventID is based on the reason for the logon failure. If the failure is due to a bad password or user name, EventID 529 is logged. If an account is disabled and the user tried logging on to that account, EventID 531 is logged. Sometimes the policy set on accounts can cause a logon failure, such as, the user is trying to logon during a restricted time period (EventID 530), the user account had been expired and the user tries to logon (EventID 532), a user is not allowed to logon on the computer in question (EventID 533), a user has not been granted the requested logon type on the machine (EventID 534), the account's password has been expired (EventID 535), or the account has been locked out possibly due to multiple failure logons (EventID 539).

$$NetworkLogon(user, session, domain) \; - \; EventID\ 540$$

When a user on the network logs on to a resource, e.g., a printer or a shared folder, this event appears in the log. The logon type is always 3 indicating a network logon, hence it's not included with the fields.

$$UserLogoff(user, session, typeOfLogon) \; - \; EventID\ 538$$

This event is logged for any account logon, whether it is for an interactive user logon or a network logon to connect with a shared folder on the network. It should be noted that if a computer is turned off before an account is logged off, Windows logging system might not log it until the computer restarts, so it's not out of the normal to see a logoff far down the log. That is why it is important to track a logoff using the session ID.

$$WinstationSessionReconnect(user, session, domain, clientName, clientAddress) \\ - EventID\ 682$$

Windows logs this event in case a a user reconnects to a disconnected terminal server session. EventID 528 will only log a new session, but does not log the event in the case of reconnection. The *user* field indicated the account used to log on, however the clientName indicated the computer name from which the user logged on.

$$WinstationSessionDisconnected(user, session, domain, clientName, clientAddress) \\ - EventID\ 683$$

Windows logs this event when a user disconnects from a terminal server (aka remote desktop) session as opposed to an full logoff which triggers event 538.

Account Management

$$AccountCreated(user, session, newAccountName, newAccountDomain) \ - \ EventID\ 624$$
$$AccountChanged(user, session, targetAccount, targetDomain, accountChange) \\ - EventID\ 642$$
$$PasswordChange(user, session, targetAccount, targetDomain, sORf) \ - \ EventID\ 627$$

When a user account is created, three events will be logged. The first is AccountCreated, followed by an AccountChanged Event with the *accountChange* field being "account created", then a Password-Change event if a password was set. The *user* field is the account from which the new user account was created and the *session* indicates the logon session ID. Needless to say, *user* and *session* fields in the three events should be the same. The *newAccountName* and *newAccountDomain* in the AccountCreated event should be the same as the *targetAccount* and *targetDomain* fields in the AccountChanges and PasswordChange events. It should be noted that if the AccountCreated and AccountChanged events appear in the log, this means that the event was successful. This is not the case for PasswordChange event which can be either successful or a failure, hence the $sORf$ field.

$$AccountEnabled(user, session, targetAccount, targetDomain) \; - \; EventID\ 626$$
$$AccountDisabled(user, session, targetAccount, targetDomain) \; - \; EventID\ 629$$
$$AccounDeleted(user, session, targetAccount, targetDomain) \; - \; EventID\ 630$$
$$PasswordSet(user, session, targetAccount, targetDomain) \; - \; EventID\ 628$$

The difference between a PasswordSet event and a PasswordChange event is that the former event indicated a password rest, which does not require the knowledge of the current password. A PasswordChange event is logged in two cases only, when a new account is created, or when a user is changing his own password. Account enabled, disabled, and deleted are self-explanatory, however, some versions of windows might not log them despite of Windows documentation. These events are logged effectively in Windows 2003 server and after.

$$AccountLockedOut(user, session, targetAccount, targetDomain) \; - \; EventID\ 644$$

An account can be locked out due to multiple reasons, one of which is consecutive failed logon attempts. This event is followed by an AccountChanged event with the *accountChange* field being "account locked".

$$GroupCreated(user, session, groupName, groupDomain) \; - \; EventID\ 635,\ 648,\ or\ 653$$
$$GroupDeleted(user, session, groupName, groupDomain) \; - \; EventID\ 634,\ 638,\ or\ 652$$
$$GroupMemberChange(user, session, groupMember, groupName, groupDomain, \\ addedORremoved) \; - \; EventID\ 636,\ 637,\ 650,\ or 651$$
$$GroupChanged(user, session, groupName, groupDomain) \\ - \; EventID\ 639,\ 641,\ 649,\ or\ 654$$

Detailed Tracking

$$ProcessCreated(user, session, processName, processID, createdBy) \; - \; EventID\ 592$$
$$ProcessExited(user, session, processName, processID) \; - \; EventID\ 593$$
$$ProcessTokenAssignment(user, assigningProcess, newProcessName, newProcessID, \\ sORf) \; - \; EventID\ 600$$

When a program executed, a ProcessCreated event is logged. This process has to run using the privileges of the account it operates under, so a token is assigned to that process. The token contains

Type of Service	Value	Description
SERVICE_WIN32_OWN_PROCESS	0x00000010	A Microsoft Win32 service that runs its own process.
SERVICE_WIN32_SHARE_PROCESS	0x00000020	A Win32 service that shares a process.
SERVICE_INTERACTIVE_PROCESS	0x00000100	A Win32 service that interacts with the desktop. This value cannot be used alone and must be added to one of the two previous types. The StartName column must be set to LocalSystem or null when using this flag.

Table 12: Service Type

Type of Service	Value	Description
SERVICE_AUTO_START	0x00000002	A service start during startup of the system.
SERVICE_DEMAND_START	0x00000003	A service start when the service control manager calls the StartService function.
SERVICE_DISABLED	0x00000004	Specifies a service that can no longer be started.

Table 13: Service Start Type

the privileges authorized for the process and is passed from one process to another. This is reflected through the ProcessTokenAssignment event. The process that assigns the token, *assigningProcess*, is the process that created the new process. The *processName* is the name and path of the process, while the *processID* is the process identifier. The same process can be executed multiple times concurrently, the *processID* is used to differentiate the different runs of the process.

$$InstallService(user, session, serviceName, serviceFile, serviceType, serviceStartType, serviceAccount, sORf) \; - \; EventID\ 601$$

This event is actually an attempt to install a service, the attempt might be a failure or success, *sORf*. The *serviceName* is the name of the service as it might appear in the services control manager, while the *serviceFile* is the complete path and file name of the installed service. For the *serviceType* and *serviceStartType* refer to tables 12 and 13 respectively. Finally, the *serviceAccount* is the account that the service is specified to run under.

$$ScheduledTaskCreated(user, session, fileName, command, triggers, time, targetUser) \; - \; EventID\ 602$$

This event is logged when a task is created and when changes are made to existing tasks. The field *fileName* indicates where the task definition file is stored, it's of type JOB. The *command* field is what will be executed when the task begins, the full path of the executable is shown here. The event that will trigger the task is indicated in the *trigger* field. As for the *targetUser*, this is the user under which the task will be executed.

Object Access

$$ObjectOpen(user, session, objectType, objectName, handleID, processID, access, sORf)$$
$$- EventID\ 560$$

Everything in Windows comes down to an object. The *objctType* indicates the type of object, for example, a file or a registry key. If a process began and an operation was to be done on that process, a handle *handleID* should be given to that process with the privileges requested. The *processID* here indicates which process opened the object, and the *access* indicates what the process is requesting to perform on the object. However, that access actually executed on the object are logged in another event:

$$ObjectAccess(objectType, handleID, processID, access, sORf)\ -\ EventID\ 567$$

Notice that the object name does not appear in this event, however, we can track the access through the *handleID*.

$$HandleClose(handleID, processID)\ -\ EventID\ 562$$

A HandleClose event is the same as an object close event. As mentioned earlier, when an object is opened, a handle is passed to it to perform whatever operations necessary. When the handle is closed, no operations can be preformed on that object until another handle is assigned.

$$ObjectOpenForDelete(user, session, objectName, objectType, handleID, processID)$$
$$- EventID\ 563$$
$$ObjectDeleted(handleID, processID)\ -\ EventID\ 564$$

To delete an object there are two possible methods, the ObjectOpenForDelete and ObjectDeleted. In the case of the former, the *FILE_DELETE_ON_CLOSE* flag is set and this is the only way to delete objects opened exclusively by another program, when installing an application for example. The ObjectDeleted event is the more common way that an object would be deleted, and the event is logged when the object is actually deleted. In the case of ObjectDeleted event, there should appear an ObjectOpen event prior to the ObjectDeleted event. If a user wants to delete a file, in most cases she will select the object and hit the delete button. In terms of Windows processes, ObjectOpen event followed by ObjectDeleted.

Policy Change

$$UserRightsAssigned(user, session, assignedTo, rights) \;-\; EventID\ 608$$
$$UserRightsRemoved(user, session, removedFrom, rights) \;-\; EventID\ 609$$

These two events indicate changes to user rights. The *assignedTo* and *removedFrom* fields show which user account has been modified, and the *rights* field indicates the rights that have been assigned or removed from the user. If these events appear in the log, this indicated that the operation was successful, it is not logged otherwise.

$$AuditPolicyChange(user, session, changes) \;-\; EventID\ 612$$

When a change is made to the audit policy, the event plus the changes are logged. The *changes* field is the list of changes made. If a policy was added, a plus sign will appear before the policy. Likewise, if a policy was removed, a minus sign will appear in front of the policy. For example, $changes\{++Logon/Logoff, --ObjectAccess, --AccountManagment, ...\}$.

$$SysSecurityAccessGranted(user, session, accountModified, access) \;-\; EventID\ 621$$
$$SysSecurityAccessRemoved(user, session, accountModified, access) \;-\; EventID\ 622$$

These two events indicate the granting or removing logon rights such as accessing a computer from the network or logging on as a service. The UserRightAssigned event does not log such events, whereas these two events are strictly for logon rights.

$$UserAccountEnabled(user, session, targetAccountID) \;-\; EventID\ 625$$

The $targetAccountID$ here is the target user name and the domain, in the form of username\domain.

Privilege Use

$PrivilegeAssigned(assignedTo, privilege)$ – $EventID$ 576

$PrivilegeServiceCalled(user, serviceName, privilege, sORf)$ – $EventID$ 577

$PrivilegeServiceCalled(user, server, serviceName, privilege, sORf)$ – $EventID$ 577

$PrivilegeObjectOperation(user, process, privilege, sORf)$ – $EventID$ 578

Event PrivilegeServiceCalled and PrivilegeObjectOperation indicate that the user exercised the rights specified in the *privilege* field. Whereas, PriviledAssigned is triggered after a logon and indicates the rights authorized for the user, but not necessarily the rights that the user accessed.

System Events

$WindowsStartingup()$ – $EventID$ 512

$WindowsShutDown()$ – $EventID$ 513

$NotificationPackageLoaded(user, packageName)$ – $EventID$ 518

$AuthenticationPackageLoaded(user, session, packageName);$ – $EventID$ 514

The AuthenticationPackageLoaded event is logged at startup once for every authentication package loaded. An authentication package is a DLL for authenticating NTLM or Kerberos for example.

$TrustedLogonProcess(user, session, processName)$ – $EventID$ 515

A logon process is a trusted part of the operating system and handles the overall logon function for different logon methods including incoming RAS connections, RunAs, interactive logons initiated by CtrlAltDel, and network logons (as in drive mappings).

$AuditLogCleared(user, session)$ – $EventID$ 517

$SystemTimeChanged(user, session, oldTime, newTime)$ – $EventID$ 520

$ResourcesExhausted()$ – $EventID$ 516

Internal resources allocated for the queuing of audit messages have been exhausted, leading to the loss of some audits. This is due to an extremely high period of activity which prevented Windows from logging security events.

Windows Firewall

$$FirewallPolicyLoaded(user, groupPolicyApplied, interface, operationalMode,$$
$$fileAndPrintSharing, remoteDesktop, UPnP, remoteAdmin, unicastResponse,$$
$$logDroppedPackets, logSuccessConn, ICMP\_SET) \ - \ EventID\ 848$$
$$WhereICMP_SET = \{inEchoReq, inTimeStampReg, inMaskReq, inRouterReq,$$
$$outDestUnreachable, outSourceQuench, outParameterProb, outTimeExceeded,$$
$$redirect\}$$
$$WFLApplicationException(user, applicationPath, state, scope) \ - \ EventID\ 849$$
$$WFLPortException(user, name, portNumber, protocol, state, scope) \ - \ EventID\ 850$$
$$WFLApplicationORServiceListening(user, appOrServiceName, path, processID, service,$$
$$rpcService, ipVer, portNo, allowed, userNotified) \ - \ EventID\ 861$$

FirewallPolicyLoaded event displays the startup configuration of the Windows Firewall. This event appears in general at startup or when the Windows Firewall is enabled and starts or restarts. It is followed by WFLApplicationException and WFLPortException events. The WFLApplicationException event is an application which is listed as an exception in the firewall configuration, and WFLPortException is a port exception in the firewall configuration. The *state* field in these two events Specifies whether the application is enabled or disabled in the exceptions list, where as the *scope* field Specifies the conditions under which the application is processed as an exception. As for the fields for FirewallPolicyLoaded, they are described in table 14. Finally, the WFLApplicationORServiceListening event will most likely appear after these events for a service or application trying to connect through the network but is not on the exception list.

Field	Possible Values	Description
groupPolicyApplied	Yes/No	Specifies whether Group Policy is applied.
interface	All/<interface name>	Specifies the network adapter (interface) to which the settings apply.
operationalMode	On/Off	Specifies which mode Windows Firewall is in.
fileAndPrintSharing	Enabled/Disabled	Specifies whether File and Printer Sharing is enabled or disabled.
remoteDesktop	Enabled/Disabled	Specifies whether Remote Desktop is enabled or disabled.
UPnP	Enabled/Disabled	Specifies whether UPnP is enabled or disabled in the exceptions list.
remoteAdmin	Enabled/Disabled	Specifies whether Remote Assistance is enabled or disabled.
unicastResponse	Enabled/Disabled	Specifies whether Windows Firewall will allow unicast traffic that is in response to multicast or broadcast traffic through the Firewall
logDroppedPackets	Enabled/Disabled/-	Log dropped packets.
logSuccessConn	Enabled/Disabled/-	Log successful connections.
inEchoReq	Enabled/Disabled/-	Allow incoming echo request - ICMP_Set
inTimeStampReg	Enabled/Disabled/-	Allow incoming timestamp request - ICMP_Set
inMaskReq	Enabled/Disabled/-	Allow incoming mask request - ICMP_Set
inRouterReq	Enabled/Disabled/-	Allow incoming router request - ICMP_Set
outDestUnreachable	Enabled/Disabled/-	Allow outgoing dest. unreachable - ICMP_Set
outSourceQuench	Enabled/Disabled/-	Allow outgoing source quench - ICMP_Set
outParameterProb	Enabled/Disabled/-	Allow outgoing parameter problem - ICMP_Set
outTimeExceeded	Enabled/Disabled/-	Allow outgoing time exceeded - ICMP_Set
redirect	Enabled/Disabled/-	Allow redirect - ICMP_Set

Table 14: FirewallPolicyLoaded fields [6]

$WFLApplicationExceptionChange(policyOrigin, profileChanged, changeType,$
$newAppName, newPath, newState, newScope, oldAppName, oldPath, oldState,$
$oldScope) - EventID$ 851

$WFLPortExceptionChange(policyOrigin, profileChanged, changeType, interface,$
$newName, newPort, newProtocol, newState, newScope, oldName, oldPort,$
$oldProtocol, oldState, oldState) - EventID$ 852

$WFLOperationModeChange(policyOrigin, profileChanged, interface,$
$newOperationMode, OldOperationMode) - EventID$ 853

$WFLLoggingSetChange(policyOrigin, profileChanged, newLogDroppedPackets,$
$newLogSuccessfulConnections, oldLogDroppedPackets,$
$oldLogSuccessfulConnections) - EventID$ 854

$WFLICMPSetChange(policyOrigin, profileChanged, interface, newICMP\_SET,$
$oldICMP\_SET) - EventID$ 855

$WFLUnicastResponseChange(unicastResponse) - EventID$ 856

$WELRemoteAdminSetChange(policyOrigin, profileChanged, newRemoteAdminSet,$
$oldRemoteAdminSet) - EventID$ 857

$WFLGroupPolicyApplied() - EventID$ 858

$WFLGroupPolicyRemoved() - EventID$ 859

$WFLProfileChange(activeProfile) - EventID$ 860

These set of events indicate any changes made to the Windows Firewall. In most cases the name implies what the event logged means, but we will mention them briefly. WFLApplicationException-Change or WFLPortExceptionChange indicate a change in either the application or port exception list. WFLOperationModeChange event shows that there were some changes to the operational mode of windows firewall. WFLLoggingSetChange reflects a change in the logging setting of windows firewall, for example not logging successful connections to connect remotely without being noticed. WFLICMPSetChange events indicates changes made to the ICMP settings, ICMP settings are mentioned in table 14. WFLUnicastResponseChange displays a change made to the Windows Firewall: Prohibit unicast response to multicast or broadcast requests Group Policy setting. A change to either enable or disable remote administration is shown by event WFLRemoteAdminSetChange. As for WFLGroupPolicyApplied and WFLGroupPolicyRemoved, the windows firewall policy as any policy on Windows operating system can be a per-user policy or a group policy. These two events indicate if a group policy was applied or removed for the user currently logged on. Finally, the WFLProfileChange event simple indicates a change in the windows firewall profile. A profile here is defined as a set of settings and configurations.

4.2.2 Firewall Log

The structure of events in the Windows Firewall log differs from the structure of events in the Windows security log. Events are not based on EventIDs.
The prefix <FWL> will be used to denote the Windows Firewall Log when constructing the models.

$$UDPReceive(sourceIP, destinationIP, action, sourcePort, destinationPort, size)$$
$$UDPOpen(sourceIP, destinationIP, sourcePort, destinationPort, size)$$
$$UDPClose(sourceIP, destinationIP, sourcePort, destinationPort, size)$$
$$UDPDrop(sourceIP, destinationIP, sourcePort, destinationPort, size)$$
$$TCPOpen(sourceIP, destinationIP, sourcePort, destinationPort, size)$$
$$TCPClose(sourceIP, destinationIP, sourcePort, destinationPort, size)$$
$$TCPDrop(sourceIP, destinationIP, sourcePort, destinationPort, size)$$

DROP indicates a packet blocked.
OPEN indicates the normal opening of a connection using either TCP or UDP protocols.
CLOSE indicates a normal closure of a TCP connection that was opened in the firewall. This is only logged when "Log successful connections" is checked.

4.2.3 Application Log

The Windows Application log records events generated by application which are configured to write to the application log. Events stored in this log are application specific, meaning that every application logs its specific data using the chosen event ID. The number of event IDs that are the same for all applications is very limited hence it is not feasible to model every possible event in the application log. In the following we provide a short list of events based on what was recorded in our logs. For future reference, the Windows application log will be denoted by <APP>.

$ServiceStart(serviceName)$

$ApplicationHang(applicationName, hangAddress)$ – $EventID$ 1002

ApplicationHang is an error message logged in the application log due to numerous reasons. In general this means that the application is busy with some processing and is not responding anymore due to some software error, incompatibility issues, or RAM problems. The *applicationName* indicates which application stopped responding, and the *hangAddress* is the hexadecimal address of the offset.

$ApplicationStoppedUnexpectedly(applicationName, faultingModule, faultAddress)$
– $EventID$ 1000

This indicates a program stopped unexpectedly. The message contains details on which program and module stopped. A matching event with Event ID 1001 might also appear in the event log. This matching event displays information about the specific error that occurred.

4.2.4 System <SYS>

$NeworkLocationAwarness(source, action)$ – – $EventID$ 7036

$ApplicationLayerGateway(source, action)$ – – $EventID$ 7036

$ApplicationLayerGateway(source, action)$ – – $EventID$ 7035

$RemoteAccessConnection(source, action)$ – – $EventID$ 7036

$ServiceStopped(serviceName)$ – – $EventID$ 6006 (*seen on Event Log Service*)

$ServiceStarted(serviceName)$ – – $EventID$ 7039

$EventFailedToRenew(networkCardAddress, error)$ – – $EventID$ 1003

The latter event ($EventID$1003) is a warning generated by the Dynamic Host Configuration Protocol (DHCP), indicating that the DHCP Client service on the computer did not receive a response from the DHCP server to renew the computer's IP address lease. The client still has a valid IP address and periodically will try to get an address lease extension. This could be caused by network connectivity issues, a DHCP server or relay malfunction, firewall issues, or a malfunction of the computer's network interface card or driver.

$MsgSxsFunctionCallFail(source, function, usedFor, errorMsg)\ -\ EventID\ 59$

This depicts a component or manifest that could not be activated. The possible causes include: the component or manifest depends on another program or component that is not installed, the manifest contains XML content that is not valid, or the user does not have the correct permissions. In our case this event was preceded by eventID 58. Windows support does not provide any information related to this event and is apparently source dependant. However we model it as such:

$FileSyntaxError(source, file, errorPointer)\ -\ EventID\ 58$

In our logs these two events were shown as follows:

$FileSyntaxError(SideBySide, \text{“}C : \backslash ProgramFiles \backslash AppleSoftwareUpdate\backslash$
$Plugins \backslash MSIInstallPlugin.dll.Manifest\text{”}, \text{“}line2\text{”})$
$MsgSxsFunctionCallFail(SideBySide, \text{“}GenerateActivationContext\text{”}, \text{“}C : \backslash Program$
$Files \backslash AppleSoftwareUpdate \backslash Plugins \backslash MSIInstallPlugin.dll.Manifest\text{”},$
$\text{“}The\ operation\ completed\ successfully\text{”})$

EventID 58, FileSyntaxError, was not documented by Microsoft. Through our analysis we determined that EventID 58 indicates where the error is located in the Manifest file which generated a MsgSxsFunctionCallFail error event.

$AuthenticationRequestFailure(source, errorMsg)\ -\ EventID\ 40968$

This event is generated by SPNEGO which provides a Single Sign-on (SSO) capability for Windows. When this event appears in the logs it means the Security System has received an authentication request that could not be decoded. However, this is not logged as an error, just a warning. In many cases SPNEGO is used when protected objects are accessed using Internet Explorer.

$UpdatesDownloaded(source, updateList)\ -\ EventID17$

Whenever updates for Windows are downloaded an information event under the category of installation is logged. This event provides details on patches and updates for the Windows Update Agent. The *updateList* is the list of downloaded updates. However, the System Log does not indicate if the updates were actually installed. This is determined from the Security log. Correlating these two logs helps us determine if the computer is properly patched and updated. This is especially important if there is a security policy that mentions patch management.

$TimeSynchronized(timeSource) \; - \; EventID\, 35$

When the Windows Time Service is synchronized, it logs an event indicating when it has been synchronized and the time source from which it acquired the time stamp.

$UnsynchronizedTime(unsyncTime) \; - \; EventID\, 36$

This is a warning event generated by the Windows Time Service when it cannot synchronize with it's time source. When a computer cannot synchronize with its source for a period of time, it will not provide the time to requesting clients. The local computer time cannot be updated until successful communication with the time source resumes. Usually, this message does not indicate an immediate problem. However, it represents a condition that can cause problems if it continues for an extended period of time.

Chapter 5

Formal Framework

In the previous chapters we have provided a background about the different previously proposed approaches dealing with forensics and log analysis, and introduced the Windows logging system and the Windows events and processes along with the log functions we will use. Having the necessary building blocks, we now demonstrate how the logs can be modeled. Information stored in logs of a computer system is of crucial importance to gather forensic evidence of investigated actions or attacks against the system. Analysis of this information should be rigorous and credible, hence it lends itself to formal methods. We propose a model checking approach to the formalization of the forensic analysis of logs. In this chapter we begin by presenting some basic definitions, then we proceed to explain our formal framework and demonstrate its use through examples. Proof of fitness, soundness and completeness are provided in the appendix.

5.1 Definitions [1]

A multi-sorted signature Σ is a pair $< \mathbb{S}, \mathbb{F} >$, where $\mathbb{S}$ is a set of sorts and $\mathbb{F}$ is a set of operator symbols. For each operator symbol f, the function **arity** : $\mathbb{F} \rightarrow \mathbb{N}$ maps f to a natural number called the arity of f. Moreover, for any operator symbol $f \in \mathbb{F}$ of arity n, we have the functions $\mathbf{dom}(f) \in \mathbb{S}^n$ and $\mathbf{cod}(f) \in \mathbb{S}$, where $\mathbb{S}^0 = \emptyset$, $\mathbb{S}^1 = \mathbb{S}$, and $\mathbb{S}^{n+1} = \mathbb{S} \times \mathbb{S}^n$. For an operator symbol f, where $\mathbf{dom}(f) = S_1 \times S_2 \ldots \times S_n$, and $\mathbf{cod}(f) = S$, f is called an operator of sort S and we write $f : S_1 \times S_2 \ldots \times S_n \rightarrow S$. In the case where $\mathbf{arity}(f) = 0$, and $\mathbf{cod}(f) = S$, f is called a constant symbol of sort S, the set of constants of sort S is written $\mathbb{C}_S$. Also, for each sort S, we define an infinite set $\mathbb{X}_S$ of countable elements called variables of sort S, such that $\mathbb{X}_S \cap \mathbb{C}_S = \emptyset$. For a certain signature Σ, we define the set $\mathbb{T}_\Sigma(S, X)$ of free terms of sort S inductively as follows (the symbol

'⇒' is for logic implication):

$$
\begin{aligned}
&(\mathbb{X}_S \cup \mathbb{C}_S) \subseteq \mathbb{T}_\Sigma(S, X) \\
&f : S_1 \times S_2 \ldots \times S_n \to S \Rightarrow f(t_1, \ldots t_n) \in \mathbb{T}_\Sigma(S, X) \\
&\text{where each } t_i \text{ is a term of sort } S_i
\end{aligned}
$$

The set $\mathbb{T}_\Sigma(X)$ of free terms over the signature Σ is defined as $\bigcup_{S \in \mathbb{S}} \mathbb{T}_\Sigma(S, X)$. The set of ground terms $\mathbb{T}_\Sigma \subseteq \mathbb{T}_\Sigma(X)$ includes all terms that do not contain any variables.

An algebra $\mathcal{A}$ of a signature Σ (called a Σ-algebra) is a pair $< \mathbb{A}, \mathbb{F}_\mathcal{A} >$. The Σ-algebra $\mathcal{A}$ assigns to each sort S in the signature Σ a set $\mathbb{A}_S$ called the carrier of sort S, where $\mathbb{A} = \bigcup_{S \in \mathbb{S}} \mathbb{A}_S$. Also for each operator symbol $f : S_1 \times S_2 \ldots \times S_n \to S$, $\mathcal{A}$ assigns a function $f_\mathcal{A} : A_{S_1} \times A_{S_2} \ldots \times A_{S_n} \to A_S$, $\mathbb{F}_\mathcal{A}$ is the set of all functions $f_\mathcal{A}$. A homomorphism is a function between Σ-algebras that reserves their structure. If $\mathcal{A}$ and $\mathcal{B}$ are two Σ-algebras having the same signature, $h : \mathcal{A} \to \mathcal{B}$ is a called a Σ-homomorphism iff:

$$\forall f \in \mathbb{F} \,.\, h(f_\mathcal{A}(a_1, a_2, \ldots a_n)) = f_\mathcal{B}(h(a_1), h(a_2), \ldots h(a_n))$$

A Σ-algebra provides an interpretation for a certain signature, where each sort is interpreted as a set and each operator symbol as a function. The free term algebra $\mathcal{T}_\Sigma(X)$ associated with a signature Σ is a special kind of Σ-Algebra in which each carrier set $\mathbb{A}_S$ of the sort S and each function $f_{\mathcal{T}_\Sigma(X)} : \mathbb{A}_{S_1} \times \mathbb{A}_{S_2} \ldots \times \mathbb{A}_{S_n} \to \mathbb{A}_S$ are defined as:

$$
\begin{aligned}
&\forall t \in \mathbb{T}_\Sigma(S, X) \Leftrightarrow t \in \mathbb{A}_S \\
&f_{\mathcal{T}_\Sigma(X)}(t_1, t_2, \ldots t_n) = f(t_1, t_2, \ldots t_n) \\
&\text{where, } t_i \in \mathbb{A}_{S_i}\text{, i.e., each term is taken to be its own interpretation}
\end{aligned}
$$

The term algebra $\mathcal{T}_\Sigma$ can be obtained from $\mathcal{T}_\Sigma(X)$ by removing any terms that contain variables. A substitution $\theta_x : \mathbb{X}_S \to \mathbb{T}_\Sigma(S, X)$ is a mapping from variables to terms of the same sort. A ground substitution maps variables to ground terms. A substitution is generally extended to a homomorphism in the following way:

$$
\begin{aligned}
&\theta_x : \mathbb{X}_S \to \mathbb{T}_\Sigma(S, X) \text{ is extended to} \\
&\tilde{\theta_x} : \mathbb{T}_\Sigma(S, X) \to \mathbb{T}_\Sigma(S, X) \\
&\tilde{\theta_x}(c) = c \text{ where } c \text{ is a constant} \\
&\tilde{\theta_x}(f(t_1, t_2, \ldots t_n)) = f(\tilde{\theta_x}(t_1), \tilde{\theta_x}(t_2), \ldots \tilde{\theta_x}(t_n))
\end{aligned}
$$

Usually we write θ_x for $\tilde{\theta_x}$. As a simple example, consider the signature $<\{\mathsf{User}, \mathsf{File}, \mathsf{Bool}\}, \{CreateFile, DeleteFile\}>$, where $CreateFile : \mathsf{User} \to \mathsf{File}$, and $DeleteFile : \mathsf{User} \times \mathsf{File} \to \mathsf{Bool}$. Now suppose we have the constants u_1, u_2 of the sort User and l_1, l_2 of the sort File, so we can have the terms $CreateFile(u_1)$, $DeleteFile(u_1, CreateFile(u_1))$ [1]

5.2 Formal Model

In this section, we propose a model for logs that will be used for our analysis. Analyzing the model amounts to a model checking problem where we express desired model properties using a logic that will be detailed below. The tableau-based proof system [1] of the logic can be further developed into a model checking algorithm. We view a log as a sequence of log entries, each entry consists of a certain term $t \in \mathbb{T}_\Sigma$ [1]. The signature Σ is chosen such that each event monitored by the logging system can be represented as a term $t \in \mathbb{T}_\Sigma$. We define the log model L as a finite sequence over $\mathbb{T}_\Sigma$, i.e., $L \in \mathbb{T}_\Sigma^*$. In other words, the log model is a sequence that represents logged events in the system ordered temporally. Graphically, the log model is a trace whose edges are labeled by terms of $\mathbb{T}_\Sigma$. However, many systems may have more than one log, where each log is dedicated to a certain category of events. Moreover, we may be faced by situations where it is necessary to inspect logs from different machines. For instance, in a computer system, one log can monitor operating system events such as file backup while another log can monitor security related events. We assume that, in the presence of more than one log in the system, all logs are handled in parallel, i.e., there is no synchronization between logs. In order to model a log system consisting of more than one log, and in the presence of concurrent actions in the logs, the model becomes a tree. In order to illustrate this, assume we have a sequence of actions $a.b$ in one log and another sequence $c.d$ in another. Temporally, we assume that a occurred first then c, then b and d occurred at the same time. The combined trace of the two logs will be a tree having the following traces $\{a, a.c, a.c.b, a.c.d, a.c.b.d, a.c.d.b\}$, i.e., we considered the two possible interleavings of the concurrent events b and d. The same idea is used to construct synchronization trees of process calculi. We define the interleaving of two logs L_1 and L_2 to be $L_1 \| L_2 \subseteq (\mathbb{T}_{\Sigma 1} \cup \mathbb{T}_{\Sigma 2})^*$ which represents all possible interleavings of concurrent events from both logs. Here, we assumed that each log has its own signature. The definition can be easily generalized in the case of a finite number of logs. Defined this way, a log system of more than one log is graphically represented as a tree whose branches are labeled by terms of term algebras. Our model consists of this log tree along with a mapping **orig** that maps terms to their respective logs. For instance, in the example above, we have **orig** $(a) = L_1$. For a sequence $s \in L$, we define $s \upharpoonright \mathbb{P}$, where $\mathbb{P}$ is a set of individual logs, to be the sequence obtained from s by removing any terms t, such that **orig** $(t) \notin \mathbb{P}$. We overload the $\upharpoonright$ operator by defining $L \upharpoonright \mathbb{P}$ to be the set $\{s \upharpoonright \mathbb{P} \mid s \in L\}$. In the example above, we have: $L \upharpoonright L_1 = \{a, a.b\}$, $L \upharpoonright L_2 = \{c, c.d\}$, and $L \upharpoonright \{L_1, L_2\} = L$. We note here that $L \upharpoonright \mathbb{P}$ is also a tree [1].

5.3 Logic for Log Properties

In this section, we present a new logic for the specification of properties of the log model. The logic is based on ideas from the ADM logic [66], with some basic differences. First, ADM is trace-based while the logic we present is tree-based, therefore we can quantify existentially and universally over traces. Moreover, this gives us the opportunity to express branching-time properties. Second, the actions in ADM are atomic symbols whereas the actions in our logic have a structure since they are terms of a term algebra. The choice of ADM in the first place is motivated by the fact that is based on modal μ-calculus with its known expressive power. Most importantly, the properties we would like to express are over traces of the log tree model, ADM has all the expressive power we need for this task including counting properties [66].

5.3.1 Syntax

Before presenting the syntax of formulas, we present the concept of a sequence pattern r. A sequence pattern has the following syntax [1]:

$$r ::= \epsilon \mid a.r \mid x_r.r \qquad a ::= t \mid \lceil t \rceil \tag{1}$$

Here a represents a term in $\mathbb{T}_\Sigma(X)$ and has the form t or $\lceil t \rceil$. Intuitively, a represents a term in the log tree model, where this term is either t or a term containing t. The term $\lceil t \rceil$ is defined as t or $f(t_1, t_2, \ldots t_n)$ such that $\exists t_i \centerdot t_i = \lceil t \rceil$. Here $f \in \Sigma$ is any function symbol in the signature of the algebra. The variable x_r represents a sequence of terms of zero or any finite length, the subscript r is added to avoid confusion with variables x of the message algebra. The sets of variables and terms in r are written $var(r)$ and $trm(r)$ respectively. Moreover, for any pattern r, the symbol $r|_i$ represents the variable or move at position i of r, where $i \in \{1, \ldots n\}$.

We define the substitution $\theta_r : var(r) \rightarrow \mathbb{T}^*_\Sigma$ that maps variables x_r in a sequence pattern to sequences of terms, this is not to be confused with the substitution θ_x that maps variables inside messages into terms of $T_\Sigma(X)$.

We define the predicate $\mathbf{match}(\sigma, r, \theta_m, \theta_r)$, which is true when a sequence $\sigma = s_1.s_2 \dots s_n$ in the log tree *matches* a pattern r, ϵ is the empty sequence:

$$\begin{aligned}
&\mathbf{match}(\epsilon, \epsilon, \theta_m, \theta_r) = true \\
&\mathbf{match}(\sigma, \epsilon, \theta_m, \theta_r) = false \qquad \text{if } s \neq \epsilon \\
&\mathbf{match}(\sigma, a.r, \theta_m, \theta_r) = (s_1 = a\theta_m) \wedge \mathbf{match}(s_2 \dots s_n, r, \theta_m, \theta_r) \\
&\mathbf{match}(\sigma, x_r.r, \theta_m, \theta_r) = \exists j \leq n \centerdot (x_r\theta_r = s_1 \dots s_j) \wedge \mathbf{match}(s_{j+1} \dots s_n, r, \theta_m, \theta_r)
\end{aligned}$$

A substitution $\theta : \mathcal{R} \rightarrow \mathbb{T}_\Sigma^*$ from patterns to sequences of terms, where $\theta = \theta_m \cup \theta_r$, is defined as follows:

$$\begin{aligned}
\theta(\epsilon) &= \epsilon \\
\theta(a.r) &= \theta_m(a).\theta(r) \\
\theta(x_r.r) &= \theta_r(x_r).\theta(r)
\end{aligned}$$

In the above defintions, we follow the usual notation for substitutions and write $r\theta$ for $\theta(r)$. From the definitions of of the predicate **match** and the substitution θ above, we notice that the condition for a match between a pattern and a sequence is the existence of one or more substitutions θ, we can therefore write the predicate as $\mathbf{match}(\sigma, r, \theta)$.

The syntax of a formula ϕ is expressed by the following grammar:

$$\varphi ::= Z \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid [r_1 \looparrowright r_2]\varphi \mid \nu Z.\varphi \tag{2}$$

We require the following two syntactic conditions:

- In $[r_1 \looparrowright r_2]$, $\forall i \centerdot (r_1|_i \in var(r_1) \Leftrightarrow r_1|_i = r_2|_i) \wedge (r_1|_i \in trm(r_1) \Leftrightarrow r_1|_i = r_2|_i \vee r_2|_i = \circledast)$.
- In $\nu Z.\varphi$, any free Z in φ appears under the scope of an even number of negations.

The first condition above means that r_2 is obtained from r_1 by replacing some of the terms of r_1 by the dummy symbol $\circledast$, where $\theta(\circledast) = \circledast$. This condition is necessary to ensure that $r_1\theta$ and $r_2\theta$ have the same length. Hence, we can replace $r_1\theta$ by $r_2\theta$ and still get a tree, which is written $L' = L[r_2\theta/r_1\theta]$, where L is the tree model of the log. The second condition is necessary for the semantic interpretation function as will be explained in the semantics section.

Intuitively, the formula $[r_1 \looparrowright r_2]\varphi$ is true if there is a sequence in the log tree that matches r_1 and when modified to match r_2 will satisfy φ. The rest of the formulas have their usual meaning in modal μ-calculus [67].

5.3.2 Semantics

A formula in the logic is interpreted over a log tree. Given a certain log tree L, a substitution θ, and a an environment e that maps formulae variables to sequences in L, the semantic function $[\![\varphi]\!]_e^{\mathcal{G}}$ maps a formula φ to a set of sequences $S \subseteq L$ [1].

$$\begin{array}{rcl} [\![Z]\!]_e^{\mathcal{G}} & = & e(Z) \\ [\![\neg\varphi]\!]_e^{\mathcal{G}} & = & L \setminus [\![\varphi]\!]_e^{\mathcal{G}} \\ [\![\varphi_1 \wedge \varphi_2]\!]_e^{\mathcal{G}} & = & [\![\varphi_1]\!]_e^{\mathcal{G}} \cap [\![\varphi_2]\!]_e^{\mathcal{G}} \\ [\![[r_1 \looparrowright r_2]\varphi]\!]_e^{\mathcal{G}} & = & \{\sigma \in L \mid \forall\theta \,.\, \mathbf{match}(\sigma, r_1, \theta) \Rightarrow \sigma' \in [\![\varphi]\!]_e^{L'}\} \\ \multicolumn{3}{l}{\text{where } \sigma' = r_2\theta \text{ and } L' = L[r_2\theta/r_1\theta]} \\ [\![\nu Z.\varphi]\!]_e^{\mathcal{G}} & = & \bigcup\{S \subseteq L \mid S \subseteq [\![\varphi]\!]_{e[Z\mapsto S]}^{L}\} \end{array} \quad (3)$$

From the semantic equations above it can be seen that the meaning of the recursive formula $\nu Z.\varphi$ is taken to be the greatest fixpoint of a function $f : 2^L \to 2^L$, where $f(S) = [\![\varphi]\!]_{e[Z\mapsto S]}^{L}$. The function f is defined over the lattice $(2^L, \subseteq, \cup, \cap)$, the syntactic condition on φ (X appears under the scope of an even number of negations) ensures that $f(S)$ is monotone [67] and hence has a greatest fixpoint.

We use the following shorthand notations:

$$\begin{array}{rcl} \neg(\neg\varphi_1 \wedge \neg\varphi_2) & \equiv & \varphi_1 \vee \varphi_2 \\ \neg\varphi_1 \vee \varphi_2 & \equiv & \varphi_1 \Rightarrow \varphi_2 \\ \neg[r_1 \looparrowright r_2]\neg\varphi & \equiv & \langle r_1 \looparrowright r_2\rangle\varphi \\ \neg\nu Z.\neg\varphi[\neg Z/Z] & \equiv & \mu Z.\varphi \end{array}$$

We also define $\nu Z.Z$ to be $\mathtt{tt}$, where $[\![\mathtt{tt}]\!]_e^{\mathcal{G}} = L$ and $\mu Z.Z$ to be $\mathtt{ff}$, where $[\![\mathtt{ff}]\!]_e^{\mathcal{G}} = \emptyset$. In the following, we prove some important results regarding the logic.

Lemma 5.3.1 $[\![\varphi[\psi/Z]]\!]_e^{\mathcal{G}} = [\![\varphi]\!]_{e[Z\mapsto [\![\psi]\!]_e^{\mathcal{G}}]}^{L}$

The proof is done by structural induction over φ.

proof

Base case: $\varphi = Z$

$[\![\psi/Z]\!]_e^{\mathcal{G}} = [\![\psi]\!]_e^{\mathcal{G}}$

But: $[\![Z]\!]_e^{\mathcal{G}} = e(Z)$, so $[\![\psi/Z]\!]_e^{\mathcal{G}} = [\![Z]\!]_{e[Z\mapsto [\![\psi]\!]_e^{\mathcal{G}}]}^{L}$

We demonstrate two cases and the other cases can be easily proved:

Case: $\varphi = \nu Z.\varphi'$ and X is free in φ

$[\![\varphi[\psi/X]]\!]^{\mathcal{G}}_e = \bigcup\{S \subseteq L \mid S \subseteq [\![\varphi'[\psi/X]]\!]^L_{e[Z\mapsto S]}\}$

By induction hypothesis:

$[\![\varphi[\psi/X]]\!]^{\mathcal{G}}_e = \bigcup\{S \subseteq L \mid S \subseteq [\![\varphi']\!]^L_{e[Z\mapsto S][X\mapsto [\![\psi]\!]^L_{e[Z\mapsto S]}]}\}$

Since ψ does not contain Z as free variable, which can be assured by renaming of the bound variable Z, we have:

$[\![\varphi[\psi/X]]\!]^{\mathcal{G}}_e = \bigcup\{S \subseteq L \mid S \subseteq [\![\varphi'[\psi/X]]\!]^L_{e[Z\mapsto S][X\mapsto [\![\psi]\!]^{\mathcal{G}}_e]}\}$

$[\![\varphi[\psi/X]]\!]^{\mathcal{G}}_e = [\![\varphi]\!]^L_{e[X\mapsto [\![\psi]\!]^{\mathcal{G}}_e]}$

Case: $\varphi = [r_1 \looparrowright r_2]\varphi'$

$[\![\varphi[\psi/Z]]\!]^{\mathcal{G}}_e = \{\sigma \in L \mid \forall\theta\,.\,\mathbf{match}(\sigma, r_1, \theta) \Rightarrow \sigma' \in [\![\varphi'[\psi/Z]]\!]^{L'}_e\}$

By induction hypothesis:

$[\![\varphi[\psi/Z]]\!]^{\mathcal{G}}_e = \{\sigma \in L \mid \forall\theta\,.\,\mathbf{match}(\sigma, r_1, \theta) \Rightarrow \sigma' \in [\![\varphi']\!]^{L'}_{e[Z\mapsto [\![\psi]\!]^{\mathcal{G}}_e]}\}$

$[\![\varphi[\psi/Z]]\!]^{\mathcal{G}}_e = [\![\varphi]\!]^L_{e[Z\mapsto [\![\psi]\!]^{\mathcal{G}}_e]}\}$ $\square$

As a result, we have $[\![\nu Z.\varphi]\!]^{\mathcal{G}}_e = [\![\varphi[\nu Z.\varphi/Z]]\!]^{\mathcal{G}}_e$. This follows from the fact that $[\![\nu Z.\varphi]\!]^{\mathcal{G}}_e = [\![\varphi]\!]^L_{e[Z\mapsto T]}$, where $T = \bigcup\{S \subseteq L \mid S \subseteq [\![\varphi]\!]^L_{e[Z\mapsto S]}\} = [\![\nu Z.\varphi]\!]^{\mathcal{G}}_e$.

We can now prove that the semantics of the expression $\mu Z.\varphi$ defined earlier as $\neg\nu Z.\neg\varphi[\neg Z/Z]$ is the least fixpoint of the function $f(S) = [\![\varphi]\!]^L_{e[Z\mapsto S]}$.

$$\begin{aligned}
[\![\neg\nu Z.\neg\varphi[\neg Z/Z]]\!]^{\mathcal{G}}_e &= L \setminus \bigcup\{S \subseteq L \mid S \subseteq [\![\neg\varphi[\neg Z/Z]]\!]^L_{e[Z\mapsto S]}\} \\
&= L \setminus \bigcup\{S \subseteq L \mid S \subseteq L \setminus [\![\varphi]\!]^L_{e[Z\mapsto [\![\neg Z]\!]^L_{e[Z\mapsto S]}]}\} \\
&= L \setminus \bigcup\{S \subseteq L \mid S \subseteq L \setminus [\![\varphi]\!]^L_{e[Z\mapsto L\setminus S]}\}
\end{aligned}$$

For any set of sequences $S \subseteq L$, let $S^c = L \setminus S$. By De Morgan laws, for any two sets A and B: $(A \cap B)^c = A^c \cup B^c$, $(A \cup B)^c = A^c \cap B^c$, $A \subseteq B \Rightarrow B^c \subseteq A^c$.

$$\begin{aligned}
[\![\neg\nu Z.\neg\varphi[\neg Z/Z]]\!]^{\mathcal{G}}_e &= (\bigcup\{L \setminus S^c \subseteq L \mid S \subseteq ([\![\varphi]\!]^L_{e[Z\mapsto S^c]})^c\})^c \\
&= (\bigcup\{L \setminus S^c \subseteq L \mid [\![\varphi]\!]^L_{e[Z\mapsto S^c]} \subseteq S^c\})^c \\
&= \bigcap(\{L \setminus S^c \subseteq L \mid [\![\varphi]\!]^L_{e[Z\mapsto S^c]} \subseteq S^c\})^c \\
&= \bigcap\{S^c \subseteq L \mid [\![\varphi]\!]^L_{e[Z\mapsto S^c]} \subseteq S^c\}
\end{aligned}$$

Moreover, we investigate the semantics of the the expression $\langle r_1 \looparrowright r_2\rangle\varphi$ as defined above:

$$\begin{aligned}
[\![\langle r_1 \looparrowright r_2\rangle\varphi]\!]_e^{\mathcal{G}} &= [\![\neg[r_1 \looparrowright r_2]\neg\varphi]\!]_e^{\mathcal{G}} \\
&= \{\sigma \in L \mid \neg\forall\theta\,.\,\mathbf{match}(\sigma, r_1, \theta) \Rightarrow \sigma' \in L \setminus [\![\varphi]\!]_e^{L'}\} \\
&= \{\sigma \in L \mid \neg\forall\theta\,.\,\mathbf{match}(\sigma, r_1, \theta) \Rightarrow \neg\sigma' \in [\![\varphi]\!]_e^{L'})\} \\
&= \{\sigma \in L \mid \neg\forall\theta\,.\,\neg(\mathbf{match}(\sigma, r_1, \theta) \wedge \sigma' \in [\![\varphi]\!]_e^{L',\theta})\} \\
&= \{\sigma \in L \mid \exists\theta'\,.\,(\mathbf{match}(\sigma, r_1, \theta) \wedge \sigma' \in [\![\varphi]\!]_e^{L',\theta})\}
\end{aligned}$$

In the derivation above we used the sequent $\psi \Rightarrow \neg\varphi \vdash \neg(\psi \wedge \varphi)$, which can be easily proved by propositional calculus. We also used the fact that for any set of sequences S, $S \cap (L \setminus S) = \emptyset$. It is worth noting here that the semantics of $\langle r_1 \looparrowright r_2\rangle\varphi$ is consistent with the definition of the modality $\langle\,\rangle$ from modal μ-calculus.

5.3.3 Tableau-based Proof System

Before we present the rules of the tableau, we define the immediate subformula relation [67] $\prec_I$ as:

$$\begin{array}{cc}
\varphi \prec_I \neg\varphi & \varphi \prec_I [r_1 \looparrowright r_2]\varphi \\
\varphi_i \prec_I \varphi_1 \wedge \varphi_2 \;\; i \in \{1,2\} & \varphi \prec_I \nu Z.\varphi
\end{array}$$

We define $\prec$ to be the transitive closure of $\prec_I$ and $\preceq$ to be its transitive and reflexive closure. A tableau based proof system starts from the formula to be proved as the root of a proof tree and proceeds in a top down fashion. In every rule of the tableau, the conclusion is above the premises. Each conclusion of a certain rule represents a node in the proof tree, whereas the premises represent the children to this node. In our case, the proof system proves sequents of the form $H, b \vdash \sigma \in \varphi$, which means that under a set H of hypotheses and the symbol b, then the sequnce σ satisfies the property φ. The set H contains elements of the form $\sigma : \nu Z.\varphi$ and is needed for recursive formulas. Roughly, the use of H is to say that in order to prove that a sequence σ satisfies a recursive formula φ_{rec}, we must prove the following: Under the hypothesis that σ satisfies φ_{rec}, then σ also satisfies the unfolding of φ_{rec}. We also define the set $H \upharpoonright \nu Z.\varphi = \{\sigma \in L \mid \sigma : \nu Z.\varphi \in H\}$. The use of H, and b will be apparent after we state the rules of the proof system:

$$\begin{array}{lcr}
R_{\neg} & \dfrac{H, b \vdash \sigma \in \neg\varphi}{H, \neg b \vdash \sigma \in \varphi} & \\
R_{\wedge} & \dfrac{H, b, \vdash \sigma \in \varphi_1 \wedge \varphi_2}{H, b_1 \vdash \sigma \in \varphi_1 \qquad\qquad H, b_2 \vdash \sigma \in \varphi_2} & b_1 \times b_2 = b \\
R_{\nu} & \dfrac{H, b \vdash \sigma \in \nu Z.\varphi}{H' \cup \{\sigma : \nu Z.\varphi\}, b \vdash \sigma \in \varphi[\nu Z.\varphi / Z]} & \sigma : \nu Z.\varphi \notin H \\
R_{[]} & \dfrac{H, b \vdash \sigma \in [r_1 \looparrowright r_2]\varphi}{\xi_1 \quad \xi_2 \quad \ldots \xi_n} & \mathit{Condition}
\end{array}$$

$$\begin{aligned}
\text{Where,}\quad & H' = H \setminus \{\sigma' : \Gamma \mid \nu Z.\varphi \prec \Gamma\} \\
& \xi_i = H, b_i \vdash r_2\theta_i \in \varphi \\
& \mathit{condition} = \begin{cases} & \forall\theta_i \,.\ \textbf{match}\ (\sigma, r_1, \theta_i) \\ \wedge & b_1 \times b_2 \ldots \times b_n = b \\ \wedge & n > 0 \end{cases}
\end{aligned}$$

The first rule concerns negation of formulas where $b \in \{\epsilon, \neg\}$ serves as a "memory" to remember negations, in this case $\epsilon\varphi = \varphi$. We define $\epsilon\epsilon = \epsilon$, $\epsilon\neg = \neg\epsilon = \neg$, and $\neg\neg = \epsilon$. Moreover, we define $\epsilon \times \epsilon = \epsilon$, $\epsilon \times \neg = \neg \times \epsilon = \neg$, and $\neg \times \neg = \neg$. The second rule says that in order to prove the conjunction, we have to prove both conjuncts. The third rule concerns proving a recursive formulas, where the construction of the set H, via H', ensures that the validity of the sequent $H, b \vdash \sigma \in \nu Z.\varphi$ is determined only by subformulas of φ [67]. The fourth rule takes care of formulas matching sequences to patterns. Starting from the formula to be proved at the root of the proof tree, the tree grows downwards until we hit a node where the tree cannot be extended anymore, i.e., a leaf node. A formula is proved if it has a successful tableau, where a successful tableau is one whose all leaves are successful. A successful leaf meets one of the following conditions:

- $H, \epsilon \vdash \sigma \in Z$ and $\sigma \in [\![Z]\!]^{\mathcal{G}}_e$.
- $H, \neg \vdash \sigma \in Z$ and $\sigma \notin [\![Z]\!]^{\mathcal{G}}_e$.
- $H, \epsilon \vdash \sigma \in \nu Z.\varphi$ and $\sigma : \nu Z.\varphi \in H$.
- $H, \epsilon \vdash \sigma \in [r_1 \looparrowright r_2]\varphi$ and $\{\sigma \in L \mid \exists\theta \,.\ \textbf{match}\ (\sigma, r_1, \theta)\} = \emptyset$.

In the next section, we prove that any formula φ has a finite tableau and that the proof system is sound and complete.

5.3.4 Properties of Tableau System

We would like to prove three main properties, namely the finiteness of the tableau for finite models, the soundness, and the completeness [1]. Soundness and completeness are proved with respect to a relativized semantics that takes into account the set H of hypotheses. The proofs are shown in Appendix: Proofs. The new semantics is the same as the one provided above for all formulas except for recursive formulas where is it defined as:

$$[\![\nu Z.\varphi]\!]_e^{L,H} = (\nu [\![\varphi]\!]_{e[Z \mapsto S \cup S']}^{L,H}) \cup S'$$
$$\text{where, } S' = H \upharpoonright \nu Z.\varphi$$

In the equation, the greatest fixpoint operator is applied to a function $f(S) = [\![\varphi]\!]_{e[Z \mapsto S]}^{L,\theta}$ whose argument is $S \cup S'$. Since the function is monotone over a complete lattice, as mentioned earlier, then the existence of a greatest fixpoint is guaranteed. We now list some results regarding the proof system. The detailed proofs are provided in the appendix.

Theorem 5.3.1 Finiteness. *For any sequent $H, b \vdash \sigma \in \varphi$ there exists a finite number of finite tableaux.*

The idea of the proof is that for any formula at the root of the proof tree we begin applying the rules $R_\neg$, $R_\wedge$, $R_{[]}$, and R_ν. The application of the first three rules results in shorter formulas, while the application of the $R\nu$ results in larger hypothesis sets H. The proof shows that shortening a formulas and increasing the size of H cannot continue infinitely. Hence no path in the tree will have infinite length. Branching happens in the proof tree whenever we have an expression of the form $\varphi_1 \wedge \varphi_2$ or $[r_1 \nrightarrow r_2]\varphi$. Finite branching is guaranteed in the first case by the finite length of any expression and in the second case by the finiteness of the model.

Theorem 5.3.2 Soundness. *For any sequent $H, b \vdash \sigma \in \varphi$ with a successful tableau, $\sigma \in [\![\varphi]\!]_e^{L,H}$*

The idea behind the proof is to show that all the successful leaves described above are valid and that the application of the rules of the tableau reserves semantic validity.

Theorem 5.3.3 Completeness. *If for a sequence $\sigma \in L$, $\sigma \in [\![\varphi]\!]_e^{L,H}$, then the sequent $H, b \vdash \sigma \in \varphi$ has a successful tableau.*

The proof relies on showing that we cannot have two successful tableaux for the sequents $H, b \vdash \sigma \in \varphi$ and $H, b \vdash \sigma \in \neg\varphi$.

5.4 Modeling Traces

Now, after we have provided the fundamental concepts, we can present our event traces. The traces presented here are a sample of the traces mostly used, and using them we can build other traces. In the following, we identify a set of sample processes such as start-ups, locking/unlocking a session, and changing a policy. We first present the steps required, through functions, for each process to take place. The return value of the functions is of type boolean, returning either true or false. Given these functions we show how they can be used to model traces from multiple log files. The functions/steps required for each process will be discussed prior to presenting the functions. More processes could be found in the Appendix section of this thesis.

To begin with, we define a list of sorts which will be used in the algebra to specify details for the events/functions in concern, table 15.

Initializing Services at Startup

When a service starts, as mentioned previously, it has to start under an account to use its privileges. What should appear in the logs is a successful log on, followed by a privilege assignment for that log on. Then the service is executed and a *ProcessBegin* appears in the security log, followed by a service start which can be logged either in the application log, or the system log, depending on the service. For the complete listing of this process please refer to Appendix: Initializing Services at Startup.

$$
\begin{aligned}
&< SEC > \mathit{SuccessfulLogon}(\mathit{NT\ Authority} \setminus \mathit{LocalService}, \mathit{sess_2Advapi}, 5)\\
&< SEC > \mathit{PrivilegeAssigned}(\mathit{NT\ Authority} \setminus \mathit{LocalService}, \{\mathit{SeAuditPrivilege},\\
&\qquad\qquad \mathit{SeAssignPrimaryTokenPrivilege}, \mathit{SeChangeNotifyPrivilege}\}, s)\\
&< SEC > \mathit{ProcessCreated}(\mathit{NT\ Authority} \setminus \mathit{System}, \mathit{sess_1}, \mathit{serviceName}, \mathit{pid_x}, c: \setminus \mathit{windows} \setminus\\
&\qquad\qquad \mathit{system32} \setminus \mathit{services.exe})\\
&< SEC > \mathit{ProcessTokenAssignement}(\mathit{NT\ Authority} \setminus \mathit{System}, \mathit{sess_1}, c: \setminus \mathit{windows} \setminus\\
&\qquad\qquad \mathit{system32} \setminus \mathit{services.exe}, \mathit{serviceName}, \mathit{pid_x}, s)\\
&< APP > \mathit{ServiceStart}(\mathit{serviceName})
\end{aligned}
$$

Using the formal framework, we model this trace as explained above. However, the events might not occur all consecutively due to multi-threading in Windows. The order of events is what we are mainly looking for, if another unrelated event(s) interrupts sequence but the order remains the same, the signature is still valid.

Sorts	Constants	Variables
file	$f_1, f_2, \ldots, f_n$	$f_x, f_y, f_z, \ldots$
user	$u_1, u_2, \ldots, u_n$	$u_x, u_y, u_z, \ldots$
object	$o_1, o_2, \ldots, o_n$	$o_x, o_y, o_z, \ldots$
objectID	$oID_1, oID_2, \ldots, oID_n$	$oID_x, oID_y, oID_z, \ldots$
processName	$pn_1, pn_2, \ldots, pn_n$	$pn_x, pn_y, pn_z, \ldots$
processID	$pID_1, pID_2, \ldots, pID_n$	$pID_x, pID_y, pID_z, \ldots$
service	$s_1, s_2, \ldots, s_n$	$s_x, s_y, s_z, \ldots$
service_type	$st_1, st_2, \ldots, st_n$	$st_x, st_y, st_z, \ldots$
registry_key	$r_1, r_2, \ldots, r_n$	$r_x, r_y, r_z, \ldots$
operation	op_{READ}, op_{WRITE}	$op_x, op_y, op_z, \ldots$
executable	$e_1, e_2, \ldots, e_n$	$e_x, e_y, e_z, \ldots$
privilege	pv_{ADMIN}	$pv_x, pv_y, pv_z, \ldots$
Access	$ac_{N\_Allowed}, ac_{ALLOWED}$	$ac_x, ac_y, ac_z, \ldots$
port	pt_{PORT_NO}	$pt_x, pt_y, pt_z, \ldots$
protocol	$proc_{UDP}, proc_{TCP}, \ldots, u_n$	$proc_x, proc_y, proc_z, \ldots$
ipAddress	$ip_1, ip2, \ldots, ip_n$	$ip_x, ip_y, ip_z, \ldots$
sourceIP	$sip_1, sip2, \ldots, sip_n$	$sip_x, sip_y, sip_z, \ldots$
destinationIP	$dip_1, dip2, \ldots, dip_n$	$dip_x, dip_y, dip_z, \ldots$
soucePort	$sp_1, sp2, \ldots, sp_n$	$sp_x, sp_y, sp_z, \ldots$
destinationPort	$dp_1, dp2, \ldots, dp_n$	$dp_x, dp_y, dp_z, \ldots$
dateTime	$dt_1, dt_2, \ldots, dt_n$	$dt_x, dt_y, dt_z, \ldots$
logonType	$lt_{INTERACTIVE}, lt_{REMOTE}, \ldots$	$lt_x, lt_y, lt_z, \ldots$
logonID	$lID_1, lID_2, \ldots, lID_n$	$lID_x, lID_y, lID_z \ldots$
logonProcess	$lp_1, lp_2, \ldots, lp_n$	$lp_x, lp_y, lp_z \ldots$
logonSession	$sess_1, sess_2, \ldots, sess_n$	$sess_x, sess_y, sess_z \ldots$
domain	$d_1, d_2, \ldots, d_n$	$d_x, d_y, d_z \ldots$
size/type	$s\_t_{POD} \ldots$	$s\_t_x, s\_t_y, s\_t_z, \ldots$
args	$a_1, a_2, \ldots, a_n$	$a_x, a_y, a_z, \ldots$
error	$er_1, er_2, \ldots, er_n$	$er_x, er_y, er_z \ldots$
bool	Tr, Fl	$b_x, b_y, b_Z, \ldots$

Table 15: Sorts of the Algebra

Hence the event x represents any unrelated event that might interrupt the sequence as shown below:

$\langle x_1.\langle\!\langle SEC\rangle\!\rangle SuccessfulLogon(u_2, sess_2, lp_{adv}, lt_{SERVICE}).x_2.\langle\!\langle SEC\rangle\!\rangle PrivilegeAssigned(u_2, axs_{srvc}, s).$
$x_3.\langle\!\langle SEC\rangle\!\rangle ProcessCreated(u_1, sess_1, pn_{sn}, pid_x, pn_{srvc}).x_4.$
$\langle\!\langle SEC\rangle\!\rangle ProcessTokenAssignment(u_1, sess_1, pn_{srvc}, pn_{sn}, pid_x, s).x_5.$
$\langle\!\langle APP\rangle\!\rangle ServiceStart(pn_{sn}).x_6 \nrightarrow \varepsilon\rangle \mathtt{tt}$

Where:

u_1	= NT Authority\ System
u_2	= NT Authority\ Local Service
u_3	= NT Authority\ Network Service
axs_{srvc}	= {SeAuditPrivilege, SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege}
pn_{srvc}	= c:\windows\system32\services.exe
pn_{sn}	= serviceName
lp_{adv}	= ADVAPI
$lt_{SERVICE}$	= Service logon, type = 5

Note about services: if there is an event in the system log or the application log of a services being started and no process begin event in the security log for the same service at the same time then this is a suspicious occurrence. It means that the security log has been tampered with. However, this is not applicable for eventLog startup event in the system log.

Start-up

First the LSASRV.dll authentication package is loaded. NETWORK SERVICE account logs on and is assigned the required privileges. A set of authentication packages are loaded starting with Kerberos.dll and MSV1_0.dll, followed by schannel.dll which is a TCP/SSL Security Provider Library, wdigest.dll which is the Digest Authentication Protocol library and is used for HTTP and SASL (Simple Authentication Security Layer), and finally setuid.dll Winlogon and Winlogon\MsGina register with the Local Security Authority KSecDD, Kernel Security Support Provider Interface, also registers with the LSA SAM loads a notification package scecli (Security Configuration Editor Client Engine). After logon is initialized, the services begin to load and an instance of svchost.exe is created by services.exe and run under services.exe privilege (by being passed a token from services.exe). Then, depending on the type of service and the privileges that it should run under, we should see an account logon followed by the service created. The accounts that logon are the local services account and the network services account. The execution of the process is shown in Appendix: Start-up.

Clear Logs

Windows logs an event called AuditLogCleared that indicates when a user deleted the security log. The easiest and most straight forward method to clear the logs is through the Microsoft Event Viewer, otherwise one has to use a program which will clear the logs manually and not through Windows. In such a case, Windows will not log if the the events have been cleared. However, it is easy to know if the logs were cleared.

Clearing the logs through Microsoft Event Viewer first triggers the event *AuditLogCleared*. The account used to clear the logs is always *NT Authority* \ *system* the session number is a variable and in this case we use $0x0, 0x3E7$. Then a *PrivilegeObjectOperation* is triggered for the *services* process under the user *Elabeth*\ *Administrator* where *Elabeth* is the domain of the account used for our experimentation. This operation requires *SeSecurityPrivilege* which is the privilege required for reading and clearing the security log. It gives access to manage the computer auditing and to manage the security log. Therefor, in the logs we will see these two events:

$$AuditLogCleared(NT\ Authority \setminus system, \text{“}0x0, 0x3E7\text{”})$$
$$PrivilegeObjectOperation(Elabeth \setminus Administrator, services.exe, pid_{860}, SeSecurityPrivilege, s)$$

These two events are modeled through our proposed algebra as follows:

$$\langle x_1.\langle\!\langle SEC\rangle\!\rangle AuditLogCleared(u_2, sess_2).x_2. x_3.\langle\!\langle SEC\rangle\!\rangle PrivilegeObjectOperation(u_1, sess_1, pn_{srvc}, pID_{860}, Tr).x_4. \looparrowright \varepsilon\rangle\mathtt{tt}$$

Where:

u_1	= Elabeth\ Administrator
u_2	= NT Authority\ system
$sess_2$	= Logon session, in this case “$0x0, 0x3E7$”
pn_{srvc}	= c:\windows\system32\services.exe
pv_1	= SeSecurityPrivilege
pID_{860}	= Process ID given at the time the process was created
Tr	= Boolean indicating a successful event s

Session Unlocking

Logon Type 7 identifies a workstation unlock event. When a user locks the session on her workstation,

or when a password protected screen saver is triggered and the user returns to her station to continue working, she has to logon again. This event is logged as a logon attempt with logon type 7.
If a screen saver is triggered, we will see the following events preceeding the session unlock event:

$$\begin{aligned}&ProcessCreated(NTAuthority \setminus system, \text{“}0x0, 0x3E7\text{”}, c : \setminus windows\setminus\\&\qquad system32 \setminus logon.scr, pid_{2992}, winlogon.exe)\\&ProcessTokenAssignment(NTAuthority \setminus system, \text{“}0x0, 0x3E7\text{”}, c : \setminus windows\setminus\\&\qquad system32 \setminus winlogon.exe, c : \setminus windows \setminus system32 \setminus logon.scr, pid_{2992}, s)\end{aligned}$$

Right before the session unlock events we will see that the screen saver has been stopped. Since stopping the screen saver is a user initiated action, a user either presses any button or moves the mouse, we will see that the screen saver has been stopped by the user, although it was started my the system. Following this event is the session unlock process as follows:

$$\begin{aligned}&ProcessExited(Elabeth \setminus Administrator, c : \setminus windows \setminus system32\setminus\\&\qquad logon.scr, pid_{2992})\\&LogonAttempt(NT\ Authority \setminus system, administrator,\\&\qquad MICROSOFT\_AUTHENTICATION\_PACKAGE\_V1\_0, s)\\&SuccessfulLogon(Elabeth \setminus Administrator, \text{“}0x0, 0x14CCAD2\text{”}, user32, 7)\\&PrivilegeAssigned(Elabeth \setminus Administrator, \{SeChangeNotifyPrivilege,\\&\qquad SeBackupPrivilege, SeRestorePrivilege, SeDebugPrivilege\})\\&UserLogoff(Elabeth \setminus Administrator, \text{“}0x0, 0x14CCAD2\text{”}, 7)\end{aligned}$$

Upon analyzing this session unlock logon, we realize that there is a user logoff right after the logon. This does not indicate that the user actually logged off. When the user unlocks the session we will always see these four events (disregarding the stopping of the screen saver). What is yet more interesting is that the logon session assigned in this logon is different than the logon session assigned at initial logon, but after the logoff the initial logon session is used and not the new one. This is better illustrated by an example: a user comes in to her office in the morning, she turns on this computer and logs on. When he logs on, she is assigned a logon session, let's call it session 1. The user then goes to get a coffee and chats with her co-workers. When she returns to her computer, she finds that the password protected screen saver has been started. She unlocks her session and is assigned logon session 2. Now she wants to start doing her job, she opens outlook and starts going through her emails. Now in the logs, we will see outlook has been opened with session 1 and not session 2. This is simply because session 2 was only used to unlock session 1.

Modeling the session unlocking using our algebra is as shown below, note that since all the events are from the security log, we denote this by placing $\langle\!\langle SEC \rangle\!\rangle$ in the beginning for the trace:

$$\begin{aligned}&\langle\!\langle SEC \rangle\!\rangle \langle x_1.ProcessCreated(u_2, sess_2, pn_{logon}, pid_{2992}, pn_{winlogon}).x_2.\\ &\quad ProcessTokenAssignment(u_2, sess_2, pn_{winlogon}, pn_{logon}, Tr).x_3.\\ &\quad ProcessExited(u_1, pn_{logon}, pid_{2992}).x_4.LogonAttempt(u_2, u_1, lp_{MS\_AUTH}, Tr).x_5.\\ &\quad SuccessfulLogon(u_1, sess_1, lp_{user32}, 7).x_6.PrivilegeAssigned(u_1, pv_1).x_7.\\ &\quad UserLogoff(Elabeth \setminus Administrator, sess_1, lt_{SESS\_UNL}).x_8. \nrightarrow \varepsilon\rangle \mathtt{tt}\end{aligned}$$

Where:

u_1	= Elabeth\ Administrator
u_2	= NT Authority\ system
$sess_1$	= Adminisrator logon session, in this case "$0x0, 0x14CCAD2$"
$sess_2$	= System logon session, in this case "$0x0, 0x3E7$"
pn_{srvc}	= c:\windows\system32\services.exe
pn_{logon}	= c:\windows\system32\logon.scr
$pn_{winlogon}$	= c:\windows\system32\winlogon.exe
lp_{MS_AUTH}	= MICROSOFT_AUTHENTICATION_PACKAGE_V1_0 logon process
lp_{user32}	= User32 logon process
lt_{SESS_UNL}	= Logon type 7 which is seesion unlocking
pv_1	= {SeChangeNotifyPrivilege, SeBackupPrivilege, SeRestorePrivilege, SeDebugPrivilege}

Scheduling and Executing a Task

Neither of scheduling or executing a task can be determined by one event. The process of scheduling a task is logged as an explorer process creating another explorer windows. An explorer windows does not prove or disprove anything, but the events that follow narrow down the possibilities. When you schedule a task, the task needs to execute under some account, for that to happen you need to provide the password. Thus we will see a logon attempt by the system to which ever account the task should execute under. Assuming that we want to schedule a command prompt task to open under administrative privileges. So we will see a logon attempt by the system to the administrator account, followed by a network logon to the administrator account followed by a logoff. Now what task has been created is never shown. We can just deduce that there is a big likelihood that a task has been created. These events are shown below:

$ProcessCreated(Elabeth \setminus Administrator,$ "$0x0, 0x200A7$", $c: \setminus windows \setminus$
$explorer.exe, pid_{1232}, c: \setminus windows \setminus explorer.exe)$
$ProcessExited(Elabeth \setminus Administrator,$ "$0x0, 0x200A7$", $c: \setminus windows \setminus$
$explorer.exe, pid_{1232}$
$NetworkLogon(Elabeth \setminus Administrator,$ "$0x0, 0x18EC33A2$", $Elabeth)$
$PrivilegeAssigned(Elabeth \setminus Administrator, SeChangeNotifyPrivilege,$
$SeBackupPrivilege, SeRestorePrivilege, SeDebugPrivilege\})$
$UserLogoff(Elabeth \setminus Administrator,$ "$0x0, 0x18EC33A2$"$)$

Modeling the session unlocking using our algebra is as follows:

$$\langle\langle SEC\rangle\rangle \langle x_1.ProcessCreated(u_1, sess_1, pn_{explorer}, pid_{1232}, pn_{explorer}).x_2.$$
$$ProcessExited(u_1, sess_1, pn_{explorer}, pid_{1232}).x_3.NetworkLogon(u_1, sess_2, d_1).x_4.$$
$$PrivilegeAssigned(u_1, pv_1).x_5.SuccessfulLogon(u_1, sess_1, lp_{user32}, 7).x_6.$$
$$PrivilegeAssigned(u_1, pv_1).x_7.UserLogoff(u_1, sess_2).x_8. \nrightarrow \varepsilon\rangle \mathtt{tt}$$

Where:

u_1	= Elabeth\ Administrator
$sess_1$	= Administrator logon session, in this case "$0x0, 0x200A7$"
$sess_2$	= Network logon session, in this case "$0x0, 0x18EC33A2$"
$pn_{explorer}$	= c:\windows\explorer.exe
d_1	= Logon domain, in this case Elabeth wich is our testing domain
pv_1	= {SeBackupPrivilege, SeRestorePrivilege, SeDebugPrivilege,

As for task execution, we have a more accurate sequence of events especially since we can correlate with the scheduled task log. Continuing with the example given above, the scheduled task execution begins by a logon attempt by the system to the administrator account. This is followed by a successful logon to the administrator account through logon type 4 which is a bach logon used for scheduled tasks. Then the scheduled tasks, the command prompt process is created. Finally, the started task is logged in the scheduled tasks log. Note that whenever it is not indicated the events pertain to the security log.

$$LogonAttempt(NT\ Authority \setminus system, Administrator,$$
$$Microsoft\_Authentication\_Package\_V1\_0)$$
$$SuccessfulLogon(Elabeth \setminus Administrator, \text{“}0x0, 018E62458\text{”}, Advapi, 4)$$
$$PrivilegeAssigned(Elabeth \setminus Administrator, SeChangeNotifyPrivilege,$$
$$SeBackupPrivilege, SeRestorePrivilege, SeDebugPrivilege\})$$
$$ProcessCreated(NT\ Authority \setminus system, \text{“}0x0, 0x3E7\text{”}, cmd.exe, pid_{1388}, svchost.exe)$$
$$< SCH > TaskStarted(cmd.exe)$$

Modeling the session unlocking using our algebra is as follows:

$$\langle x_1.\langle\!\langle SEC\rangle\!\rangle LogonAttempt(u_1, lp_{MS\_AUTH}).x_2.$$
$$\langle\!\langle SEC\rangle\!\rangle SuccessfulLogon(u_1, sess_1, lp_{adv}, lt_{batch}).x_3.$$
$$\langle\!\langle SEC\rangle\!\rangle PrivilegeAssigned(u_1, pv_1).x_4.$$
$$\langle\!\langle SEC\rangle\!\rangle ProcessCreated(u_2, sess_2, pn_{cmd}, pid_{1388}, pn_{svchost}).x_5.$$
$$\langle\!\langle SCH\rangle\!\rangle TaskStarted(u_1, pn_{cmd}).x_6. \nrightarrow \varepsilon\rangle \mathtt{tt}$$

Where:

u_1	= Elabeth\ Administrator
u_2	= NT Authority\ system
$sess_1$	= Administrator logon session, in this case “$0x0, 018E62458$”
$sess_2$	= Network logon session, in this case “$0x0, 0x3E7$”
lp_{MS_AUTH}	= MICROSOFT_AUTHENTICATION_PACKAGE_V1_0 logon process
lp_{adv}	= ADVAPI
lt_{batch}	= Logon type 4 known as Batch logon
pn_{cmd}	= cmd.exe
$pn_{svchost}$	= svchost.exe
pv_1	= {SeChangeNotifyPrivilege, SeBackupPrivilege, SeRestorePrivilege, SeDebugPrivilege}

Windows Shut-Down

When the user hits the command to shut down his computer, the command executed requests a SeShutdownPrivilege, this appears as privilege object operation in the security log. Following this request is the user initiated logoff event. This shows us that the system was turned off and not crashed for example. In the firewall logs we should see TCP connections being closed. Following there should be at least one, if not a series, process exit event in the security log. Then Windows shut down event is logged in the security log indicating that all logon sessions are being terminated. The last event to be logged is in the system log and it indicates the Event Log Service being stopped.

$< SEC > PrivilegeObjectOperation(user, process, SeShutdownPrivilege, s)$
$< SEC > UserInitiatedLogoff(username)$
$< FWL > TCPClose(localIP, -, -, -)$
$< SEC > ProcessExited(user, sess_x, processName, pid_x)$
$< SEC > WindowsShutDown()$
$< SYS > ServiceStopped(\text{“}EventLogService\text{”})$

Note the dashes (-) in *TCPClose*, this indicated that we do not care what the value of that field is. When representing this in the algebra we use variable, as noted in table 15. Hence the Windows shut-down trace is represented as follows:

$$\begin{aligned}&\langle x_1.\langle\!\langle SEC\rangle\!\rangle PrivilegeObjectOperation(u_1, pn_1, pv_1, Tr).x_2.\\ &\langle\!\langle SEC\rangle\!\rangle UserInitiatedLogoff(u_1).x_3.\\ &\langle\!\langle FWL\rangle\!\rangle TCPClose(sip_1, dip_x, sp_x, sport_x, dport_x, s\_t_x).x_4.\\ &\langle\!\langle SEC\rangle\!\rangle ProcessExited(u_1, sess_x, pn_x, pid_x).x_5.\\ &\langle\!\langle SEC\rangle\!\rangle WindowsShutDown().x_6.\langle\!\langle SYS\rangle\!\rangle ServiceStopped(s_1).x_7. \hookrightarrow \varepsilon\rangle \mathtt{tt}\end{aligned}$$

Where:

u_1	= user	pn_1	= process
pn_x	= any process	pv_1	= {SeShutdownPrivilege}
Tr	= Boolean indicating a successful event	s_1	= Event Log Service
sip_1	= the source IP is localIP	dip_x	= any destination IP
sp_x	= any source port	dp_x	= any destination port
$sport_x$	= any source port	$dport_x$	= any destination port
s_t_x	= any size	$sess_x$	= any logon session

Chapter 6

Correlation of Registry and Log Files for Forensic Analysis

6.1 Introduction

Since Windows 95, Microsoft started storing information about the operating system and installed applications in a single repository, the registry[68]. Prior to that, Windows was using .ini files, but as more and more applications were installed, managing all the .ini files was near impossible. For example, an application's .ini file is not necessarily placed in its home folder. By introducing the registry into the operating system, Microsoft provided a solution for storing all critical system information in a centralized location. The registry became Window's hierarchial database[8] containing information such as system configurations, hardware devices, applications and services, user data and profiles, temporary data, and recently used files. The registry is composed of generally five root keys, called hives. Each hive begins with HKEY, which is an abbreviation for Handle to a Key. Of the five hives, there are two main hives which are: HKEY_Users and HKEY_Local_machine. The rest of the keys namely HKEY_Classes_Root, HKEY_Current_User, and HKEY_Current_Config, are duplicates of the two main hives used for quick access and performance improvement.

Needless to say, all the information that can be retrieved from the registry is extremely useful for a forensic investigation and even to system and security administrators. However, our main focus is how this data be correlated with logs to leverage the forensic analysis. The fact is, information gathered from the registry can be used to validate logs and provide a better understanding of the system. What we mean by validation here is that events occurring in the log files can be verified through the registry to determine if the logs have been tampered with. When a user carries out some kind of malicious activities on a system, the first thing that comes to mind is covering any incriminating traces. The easiest and most straightforward way is deleting or modifying the logs, and maybe the registry. But unless the attacker is extremely careful and knows the system inside out, she is bound to leave some footprints, especially since the registry has duplicate key entries. Even using anti-forensic tools, some evidence is left out as shown in [69]. Even when an application is uninstalled, it is very rare that the uninstall does a complete job of removing all the entries in the registry related to that application[8]. Thus correlating multiple sources of evidence can help reconstruct the attack.

Furthermore, since the registry stores systems information, we can use that fact to construct a model of the system determining regular events and procedures. Thus this suggests the use of anomaly-detection techniques, whereby, models of normal behavior are constructed and subsequently checking observed behavior against these models. The models we construct are rules and relationships such that when they are not satisfied, an alert is triggered indicating an anomaly.

As far as related work in this area, some minor work has been done on the registry, from a forensic standpoint, mainly motivating further research and the significance of the registry. A paper presented by H.Cavery [70] briefly explains the structure of the registry and gives some examples of the type of information that can be found in the registry. Similar work has been done in [71], which gives an insight to the potential use of the registry within an investigation related to the internet usage. The latter also discusses the tools that are useful for such an investigation. However, in both papers, the work presented was merely an insight. They do not dig into details, but rather state some examples and hives that can be of interest.

Another paper that instigates work on registry, and more importantly correlation, is the work presented in [69]. This work motivates registry analysis since it shows that even with counter-forensic tools some evidence remains in the registry. The author analyzes 13 commercial counter-forensic tools and shows that there is significant evidentiary data. The study presented does not revolve around the registry only, but it shows the residual data in different areas after using counter-forensic tools. In all cases the registry contained some evidence of recent usage.

There has been some work done in the area of anomaly detection using the Windows registry like in [72]. The registry was used to detect malicious software by using a host-based IDS for Microsoft Windows. The Registry Anomaly Detection (RAD) system presented monitors the accesses to the registry in realtime detecting any actions performed by malicious software. Since most system activities interact with the registry, a sensor was built on the registry and the information gathered was used to detect anomalies. The anomaly detection algorithm is a registry-specific version of PHAD (Packet Header Anomaly Detection) which was originally used to detect anomalies in packet headers. In another paper, [73], a comparative study of two algorithms for Windows registry anomaly detection was conducted. The first algorithm evaluated was the one used in [72]. The second anomaly detection algorithm evaluated uses a One-Class Support Vector Machine (OCSVM) to detect anomalous activities using different kernel functions. The evaluation shows that probabilistic anomaly detection algorithm performs better in accuracy and computational complexity over support vector machine implementation under three different kernel functions. The result of this comparative evaluation is confirmed in [74].

Following in this chapter, we discuss how the registry can be correlated with the logs and how the data gathered from the registry can leverage log analysis. In some cases we will briefly explain how the correlation can be implemented providing detailed examples to illustrate our point.

6.2 Correlation Methodology

In this section we present some correlation of the Windows registry and logs. The purpose is to demonstrate the usefulness of such a correlation and the added value it presents.

6.2.1 Determining the Audit Policy from the Registry

Before we begin analysis and correlation, we must know which logs are available on the system. Most logs are not enabled by default, even the audit policy for the security log is set to no audit by default. The audit policy can be determined from the Policy key in the security hive [7]. Administrators do not have access to this key by default since it is in the Security hive. To view this key from the registry editor, you need to change the permission to have read access. The location of this key is:

$$HKEY\_LOCAL\_MACHINE \setminus \textit{Security} \setminus \textit{Policy} \setminus \textit{PolAdtEv}$$

The key value is in the form of:

$$0Z211400\ 0A000000\ 0B000000\ 0C000000\ 0D000000\ 0E000000\ 0F000000\ 0G000000\ 07000000$$

According to Microsoft [7], the value contains seven values: A to G, each being a different policy. However, the security log has nine categories and not seven, hence the values should be A to I. After

Value	Category
A	Audit System Events
B	Audit Logon Events
C	Audit File and Object Access
D	Audit Privilege Use
E	Audit Process Tracking
F	Audit Policy Change
G	Audit Account Management
H	Audit Directory Service Access
I	Audit Account Logon Events
Z	Determines if the policy is enabled or disabled

Table 16: Audit Categories [7]

Log Type	Key	Value	Data
Scheduled Tasks	HKLM\Software\Microsoft \Scheduling Agent	LogPath	%SysRoot%\Schedlg.txt, %SysRoot%\Tasks \Schedlg.txt
IPSEC	HKLM\System\CurrentControlSet \Services\Policy\Oakley	EnableLogging	1
Time Service	HKLM\System\CurrentControlSet \Services\W32Time\Config	FileLogName	logName
SAM - Acc. Lockouts	HKLM\System\CurrentControlSet \Control\Lsa	SamLogLevel	1
LSA	HEKY\System\CurrentControlSet \Control\Lsa\Kerberos \Parameters	LogToFile	1
Windows Management Instr. (WMI)	HKLM\Software\Microsoft\WBEM \CIMOM	Logging-Directory	%SysRoot%\system32 \WBMEM\Logs

Table 17: Log Files in the Registry [7]

examining the registry, the actual value is in the form of:

$$0Z\ 21\ 14\ 00\ \ 0A\ 00\ 00\ 00\ -\ 0B\ 00\ 00\ 00\ \ 0C\ 00\ 00\ 00$$
$$0D\ 00\ 00\ 00\ \ 0E\ 00\ 00\ 00\ -\ 0F\ 00\ 00\ 00\ \ 0G\ 00\ 00\ 00$$
$$0H\ 00\ 00\ 00\ \ 0I\ 00\ 00\ 00\ -\ 07\ 00\ 00\ 00$$

The values of A, B, C, D, E, F, G, H, and I correspond to a number between 0 and 3. 0 meaning that no audit is enabled for that category. 1 specifies that only success audits are enabled to be logged. 2 means only failure audits are enabled. Lastly, 3 specifies that both audits and success are enabled for that category. The categories corresponding to each letter is in table 16.

Other than the security log, there are other logs that can be determined from the registry. Table 17 lists some of the log files that are enabled through the registry and their keys.

6.2.2 The Registry as a Log File

The registry keys have a value associated with them called the LastWrite. This is similar to the last modification time of a file. The LastWrite value can be used to determine last logons, for example, to correlate with logs to determine any signs of log tampering. Changes to the registry are actually stored in a log file. Under Windows XP the log files are stored in c:\windows\system32\config. But as most log files used by the system, they cannot be accessed directly. However, if you export the registry to a text file using regedit.exe, the last write time and date are exported with it. This can prove extremely useful if we know the exact or at least the approximate date of an incident. We can retrieve all the changes made to the registry during that period and compare it with the logs. Another way to get the LastWrite time is by using a tool such as Keytime.exe [68].

One limitation to the LastWrite time is that you can retrieve the time a key was modified, but not for the specific values. So unless the Key has one value, you cannot determine which value was modified. This is where correlation comes in handy. Assume in an investigation you find that the Run Key has been modified and during the same time there was a File Open event or an Object Open event in the security log. If that file name is a value in the Run Key, then we conclude that it was the value that has been modified.

6.2.3 System Startup

When Windows starts up, the procedures and order of execution are found in the registry. For example, which boot file to use, auto start programs, services to be initialized and executed. Using this information along with some knowledge about windows start up, we can determine exactly what should be in the log file. For example, assume the registry indicates that there are four applications and ten services that load at start-up, while the log files shows three applications started and nine services initialized, this could indicate that the missing application and service might be some kind of malware or the log file has been tampered with.

Auto Start Locations

Auto start locations are one of the first places an investigator might want to look for any malware that is set to initialize at system boot up and is running in the background without the user's knowledge. This is normally the case for services. A user might never know if a service is initialized at startup if it is set to run in the background.

The auto startup keys are the following:

$HKEY\_LOCAL\_MACHINE \setminus SOFTWARE \setminus Microsoft \setminus Windows \setminus Run$
$HKEY\_LOCAL\_MACHINE \setminus SOFTWARE \setminus Microsoft \setminus Windows \setminus Runonce$
$HKEY\_CURRENT\_USER \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion \setminus Run$
$HKEY\_CURRENT\_USER \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion \setminus Runonce$
$HKEY\_USERS \setminus .DEFAULT \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion \setminus Run$
$HKEY\_USERS \setminus S-1-5-18 \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion \setminus Run$
$HKEY\_USERS \setminus S-1-5-19 \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion \setminus Run$
$HKEY\_USERS \setminus S-1-5-20 \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion \setminus Run$
$HKEY\_USERS \setminus S-1-5-xxx \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion \setminus Run$

Services

The Services key contains a long list of installed services. However, not all of them might be running. The list of subkeys under the Services key can easily exceed 200, these are only the Windows services. The fact that services can run in the background with no need for any user intervention and no visiual indication that the service is running opens a window of opportunity for malware to operate. Correlating the Services key with the logs not only serves to ensure the integrity of both sources of evidence, but also provides a means for determining if any backdoor, for example, is loaded at startup.

The path of the Services key in the registry is:

$$HKEY\_LOCAL\_MACHINE \setminus SYSTEM \setminus CurrentControlSet \setminus Services$$

The values of any service in the registry[75] are shown in table 18.

In order to construct a startup model, first we need to get all the services with Start = 2, then we need to get all their dependencies. These services are set to load automatically and their dependencies should be reflected in the security log, system log, and the application log.

6.2.4 System Information

Time Settings

One of the main elements of concern in correlation and analysis is the time stamps. If the system time is changed, this can cause ambiguity and inaccurate analysis. Thus it is important to check

Value	Type	Data	Description
DependOnGroup	REG_MULTI_SZ		Specifies zero or more group names. If one or more groups is listed, at least one service from the named group must be loaded before this service is loaded.
DependOnService	REG_MULTI_SZ		(Same as above)
Description	REG_SZ		Service description.
DisplayName	REG_SZ		Display name of the service.
ErrorControl	REG_WORD	0	Ignore: If driver fails to load/initialize, startup proceeds, no warning message appears
		1	Normal: If driver fails to load/initialize, startup proceeds, warning message appears.
		2	Severe: If driver fails to load/initialize, declares startup as failed and restarts by using the LastKnownGood control set.
		3	Critical: If driver fails to load/initialize, declares startup as failed & restarts using LastKnownGood control set. If it's already used, stops startup & runs debugging
Group	REG_SZ		The group of which a service is a member
ImagePath	REG_EXPAND_S		Full path of executable & command to execute If this is a driver, the image name is *.SYS
ObjectName	REG_SZ		Contains account name for services or driver object that the I/O manager uses to load.
Start	REG_WORD	0	Boot: Loaded by kernel loader.
		1	System: Loaded by I/O subsystem, the driver is loaded at kernel initialization.
		2	Automatic: Loaded by the Service Control Manager
		3	Manual: The service does not start until user starts it manually
		4	Disabled: Specifies that the service should not be started
Type	REG_WORD	1	A kernel-mode device driver.
		2	A file system driver.
		4	A set of arguments for an adapter
		8	A file system driver service.
		10	A Win32 program that can be started by the Service Controller and obeys its protocol. This type of service runs in a process by itself.
		20	A Win32 service that can share a process with other Win32 services.

Table 18: Service Values in The Registry [5]

the time settings and when was the last time this was modified. The time settings can be found in the following key:

$$HKEY\_Local\_Machine \setminus System \setminus CurrentControlSet \setminus Control \setminus TimeZoneInformation \setminus bias$$
$$ActiveTimeBias\ is\ the\ number\ of\ minutes\ (+\ or\ -)\ to\ add\ to\ UTC$$
$$TimeZoneInformation\ key\ also\ holds\ data\ pertaining\ to\ any\ daylight\ saving\ quirks$$

System Crash

Since the registry maintains most of the system information, this information is useful for correlation in a sense that it provides guidelines for analysis. For example, if the Windows operating system was terminated unexpectedly, this is logged in the system log. We assume that an investigator is trying to determine the preliminary phase of an attack, and on his check list is an unexpected termination of Windows. However, if this option was not enabled, the event will never be logged. The first thing would be to confirm that this event is set to be logged. This can be confirmed from the following registry key:

$$HKEY\_LOCAL\_MACHINE \setminus SYSTEM \setminus CurrentControlSet \setminus Control \setminus CrashControl$$
$$KeyValue : LogEvent = 1$$

Environment Variables

The environment key is another key of significant importance. According to Microsoft [76], the default permissions enable the group 'Everyone' full access on the system root folder. The system root is invoked under certain conditions, for example when a user logs on. Thus a malicious user with minimal privileges can modify the system root and replace a commonly used program by a malware with the same name. When another user logs on and executes that program, the malware executes instead with the privileges of that user. Therefore, correlating the environment key:

$$HKEY\_LOCAL\_MACHINE \setminus SYSTEM \setminus CurrentControlSet \setminus Control \setminus SessionManager$$
$$\setminus Environment$$

with the events relating to that malware will clarify the origin of the attack. Furthermore, using the "LastWrite" date and time of that key, we can determine when the environment variable was modified and locate, from the security log, the user logged on at that time.

Changing The Environment Variable

When a single environment variable is changed, the entire key values pertaining to the environment variables are refreshed. This being the case, it is not possible to determine which environment variable has been modified from the registry and logs alone. However, we can still determine if any modification has been made by correlating the security log and the registry. Since all the environment variables are refreshed in the registry, the security log will record two privilege object operations (Event ID 578) for each environment variable, one to delete the value, and one to set the new value. However, we will not show all the privilege object operations, instead we will represent them by only one event for simplicity. The correlation is as follows:

Process Created(EventID 592) :

File name : *c* : \windows \\ *system*32 \\ *rundll*32.*exe*

Creator process : *explorer.exe*

User name : *Administrator*

LogonID : *x* (*x here represents a constant logonID*)

Privilege Object Operation(EventID 578) :

Object server : *SC Manager*

PID : *the PID pertains to services.exe*

User name : *Administrator*

LogonID : *x*

Privileges : *SeTakeOwnershipPrivilege*

Object Open(EventID 560) :

Object server : *Security Account Manager*

Object type : *SAM_ALIAS*

Object name : *Domains* \\ *Builtin* \\ *Aliases* \\ 0000022*B*

File name : *c* : \windows \\ *system*32 \\ *lsass.exe*

Accesses : *AddMember*, *RemoveMember*, *ListMembers*, *ReadInformation*

The reason for these accesses is to list the user environment variables.

$Handle\ Closed(EventID$ 562) :

$File\ name:$ c : \windows \\ *system32* \\ *lsass.exe*

$Privilege\ Object\ Operation(EventID$ 578) :

Object server : Security

PID : the PID pertains to rundll32.exe

User name : Administrator

LogonID : x

Privileges : SeTakeOwnershipPrivilege

Up till here, these events were only to get the current environment variables. Now to do any modification we need to get the corresponding privileges these privileges are obtained from the Network Services account

$Successful\ Logon(EventID$ 528) :

User name : Network Services

Logon type : 5 (*Service Logon*)

Logon process : Advapi

$Special\ Privileges\ Assigned\ to\ New\ Logon(EventID$ 576) :

User name : Network Services

Privileges : SeAuditPrivilege, SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege

$Process\ Created(EventID$ 592) :

File name : c : \windows \\ *system32 wbem* \\ *wmiprvse.exe*

User name : System

$Primary\ Token\ Assigned\ To\ Process(EventID$ 600) :

File name : c : \windows \\ *system32 wbem* \\ *wmiprvse.exe*

Assigning process : c : \windows \\ *system32 svchost.exe*

User name : System

Privilege Object Operation(EventID 578) :

Object server : Security

PID : the PID pertains to rundll32.exe

User name : Administrator

LogonID : x

Privileges : SeTakeOwnershipPrivilege

Registry Key Changed :

Key : HKEY_Current_User \ Environment

Registry Key Changed :

Key : HKEY_Local_Machine \ System \ CurrentControlSet \ Control\ SessionManager \ Environment

6.2.5 User Activity

NTUSER.DAT holds all the registry settings for a user as well as actions taken. The contents of this file are mapped to the HKEY_USERS\SID registry hive. This registry hive is used to create the HKEY_CURRENT_USER hive when the corresponding user logs in. However, data found in this hive is related only to the local user. If for example, the current user created another user, this will not be reflected in this hive. Such a user activity will be found in the HKEY_LOCAL_MACHINE_SAM hive.

Creating a User

When a user is created, evidence of this activity will be found in the security log and the registry. The Security Account Manager (SAM) contains users created in the local domain. Under the SAM key, there is the Domains\Accounts which contains almost everything regarding users and groups [8]. This key contains three subkeys: Aliases, Groups, and Users. Under Aliases there are subkeys for each group. Aliases\Members lists the user IDs for each of the aliases. Aliases\Names lists the names for each of the aliases. In Windows XP Groups contains only one subkey which is 000000201. This subkey relates to the None group which contains ordinary users [8]. The Users subkey lists an entry for each user defined in the SAM. The first two subkeys are created by the system and are 000001F4, administrator account, and 000001F5, the guest account. When a user is created, the new user data is added into two values, F and V. These values contain information such as group membership, privileges, passwords, and all other data specified by the administrators [8].

Another subkey in SAM key is Domains\Builtin, which contains the same subkeys as

Subkey Name	Builtin Local Group	SID
00000220	Builtin\Administrators	S-1-2-32-xxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx-544
00000221	Builtin\Users	S-1-2-32-xxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx-545
00000222	Builtin\Guests	S-1-2-32-xxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx-546
00000223	Builtin\Power Users	S-1-2-32-xxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx-547
00000224	Builtin\Account Operators	S-1-2-32-xxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx-548
00000225	Builtin\Server Operators	S-1-2-32-xxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx-549
00000226	Builtin\Print Operators	S-1-2-32-xxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx-550
00000227	Builtin\Backup Operators	S-1-2-32-xxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx-551
00000228	Builtin\Replicator	S-1-2-32-xxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx-552
0000022B	Builtin\RemoteDesktopUser	S-1-2-32-xxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx-555
0000022C	Builtin\Network Configuration Operators	S-1-2-32-xxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx-556

Table 19: Builtin Local Groups [8]

Domains\Accounts. However, the subkeys contain different data. The subkeys Groups and Users are empty, and the subkey Aliases contains the builtin local group. When an account is created and added to one of the local groups, the new account data is added (value name is C) to the corresponding group under Domains\Builtin\Aliases. Table 19 lists the builtin local groups defined in Windows XP.

Creating, deleting, or modifying a user will be carried out by the Local Security Authentication Service (lsass.exe). When a user is created, the security log will indicate an Object Open Event (Event ID 560 and object type is SAM_DOMAIN) showing that a handle has been given to lsass.exe with access to read password parameters, create user, and lookup IDs. Then a Security Enabled Global Group Member is added (Event ID 632) with member ID = Domain\NewAccountName, and then the user account is created (Event ID 624). In the registry, the following key will be created:

HKEY_LOCAL_MACHINE \ Sam \ Sam \ Domains \ Account \ Users \ NewAccountName

After the key is created, the account is enabled (Event ID 626) and changed (Event ID 642). When these two events are logged, additional registry keys are created and their values set:

HKEY_LOCAL_MACHINE \ SAM \ SAM \ Domains \ Account \ Groups \ HEX_VALUE_1

Value = C

HKEY_LOCAL_MACHINE \ SAM \ SAM \ Domains \ Account \ Users \ HEX_VALUE_2

Value1 = F

Value2 = V

HKEY_LOCAL_MACHINE \ SAM \ SAM \ Domains \ Account

Value = F

HKEY_LOCAL_MACHINE \ SAM \ SAM \ Domains \ Builtin \ Aliases \ Member
    \NewMemberSID \ HEX_VALUE_2

Value = (Default)

HKEY_LOCAL_MACHINE \ SAM \ SAM \ Domains \ Builtin \ Aliases
    \Member \ NewMemberSID

Value = (Default)

The first stage of creating the user is finished, so the handle created by lsass.exe for SAM_Domain can be closed (Event ID 562). However, creating a user takes a little more then just adding the name in the SAM. Another object is opened by lsass.exe (Event ID 560), this time the object is of type SAM_ALIAS to access Domains\Builtin\Aliases\00000221. Then a security enabled local group member is added (Event ID 636) and the handle is closed to that object (Event ID 562). As a result to this event, the following keys are modified:

HKEY_LOCAL_MACHINE \ SAM \ SAM \ Domains \ Builtin \ Aliases \ 00000221

Value = C

HKEY_LOCAL_MACHINE \ SAM \ SAM \ Domains \ Builtin \ Builtin

Value = F

HKEY_LOCAL_MACHINE \ SAM \ SAM \ Domains \ Account \ Users \ HEX_VALUE_2

Value1 = F

Value2 = V

HKEY_LOCAL_MACHINE \ SAM \ SAM \ Domains \ Builtin \ Aliases \ Member
    \NewMemberSID \ HEX_VALUE_2

Value = (Default)

For the last stage of creating a user, another object is opened by lsass.exe, object type is SAM_ALIAS, for Domains\Builtin\Aliases \00000220. As in the previous stage, a security enabled local group member is added and the handle is closed. The registry keys that are modified are:

$HKEY\_LOCAL\_MACHINE \setminus SAM \setminus SAM \setminus Domains \setminus Builtin \setminus Aliases \setminus 00000220$

$Value = C$

$HKEY\_LOCAL\_MACHINE \setminus SAM \setminus SAM \setminus Domains \setminus Builtin \setminus Builtin$

$Value = F$

Deleting a User

In the case of deleting a user, we will delete the user we just created. This process is just the reverse of creating a user. Three objects need to be opened: Domains\Account Users\HEX_VALUE_2 with access to delete, Domains\Builtin\Aliases \00000221 with access to remove member, and object Domains\Builtin\Aliases \00000220 also with access to remove member. The user must be removed from whatever groups he was added to before being deleted.

Thus the security log will show an object open event (Event ID 560), opened by lsass.exe, of type SAM_USER, to access Domains\Account Users\HEX_VALUE_2 with access delete. Then another object open event, opened by the same process, of type SAM_ALIAS to handle Domains\Builtin\Aliases\00000221 with access to remove member. Using this handle a security enabled member is removed (Event ID 637) from Builtin\Users and the handle is closed. Another object is opened also of type SAM_ALIAS to handle Domains\Builtin\Aliases\00000220 with access to remove member. Using this handle, a security enabled member is removed from Builtin \Administrators and that handle is closed. Now using the first handle that was opened a security enabled member is removed from Domain\None. At this point the user has been removed from all the groups and from the domain, now the account can be deleted (Event ID 630) and the handle is closed.

As far as the registry is concerned, we will not go into each key that is modified for the de-referencing of the user. Instead we will just mention the keys that have been deleted to remove the user.

HKEY_LOCAL_MACHINE \ SAM \ SAM \ Domains \ Builtin \ Aliases \ Member \NewMemberSID \ HEX_VALUE_2
HKEY_LOCAL_MACHINE \ SAM \ SAM \ Domains \ Account \ Users \ Names \AccountName
HKEY_LOCAL_MACHINE \ SAM \ SAM \ Domains \ Account \ Users \ HEX_VALUE_2

6.2.6 Mounted Devices

Every removable device that is mounted during the existing logon session is stored in the registry in the following key:

HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion \ Explorer \MountPoints2 \ CPC \ Volume

The sub-keys of this key contain each device mounted including USB removable disks and DVD/CDROM drives. The sub-key value, named Data, contains information specifying what kind of device was mounted along with device's GUID(Globally Unique Identifiers). Another registry that could supplement information about removable devices specifically USB devices and external memory cards is the:

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USBSTOR

This key contains all the USB devices that have been plugged into the system regardless of the session.

The importance of these two keys becomes evident when an attacker installs a backdoor, for example, from a removable device then leaves. The information from these two keys along with the LastWrite time of the corresponding keys can be correlated with the security log to prove such a scenario. If such a scenario occurred, the security log will show an Object Open event (Event ID 560) or Process Begin event (Event ID 592) with the path of the file indicating a non-existing removable device. The registry will show that at that particular time a removable device has been used. The type of device and the GUID can be also used as part of the evidence if needed. To further illustrate this, we will give an example of what is exactly modified in the registry and what is logged when a USB device is connected.

The USB key we use is a standard 256MB flash stick, so there is no advanced drivers or libraries that need to run. When the USB device is plugged in, we will see in the security log that the process rundll.exe started (Event ID 592) and the process that created it was svchost.exe. As for the registry, the number of keys created and set is quite a numerous. The list of keys can be found in Appendix: Registry Correlation: USB Mounting Related Keys.

6.2.7 Installing a Service

Anonymous services installed on the system constitute a high security risk. More importantly is determining when the service was installed and who installed it. In this section we will continue from the previous example (mounting a device) to illustrate how to determine details about services installed. From the mounted device, USB device, we will install mySQL service. The correlation in this case includes application, system, and security logs as well as the registry.

When the user runs the mySQL setup.exe file from the registry, the security log records a new process created (Event ID 592) and the file name is: \Device\Harddisk\DP(1)0-0+f\mysql\setup.exe, with the creator process being explorer.exe. Note the "f" in the file name, this indicates the drive from which the file was executed. This will be seen in the registry and will be used to further link this event with the registry. In the registry, explorer.exe sets three values in two places in the registry:

HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ ShellNoRoam \ MUICache
Value : F : \mysql \ setup.exe
HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion \ Explorer
\UserAssis \ {GUID}
Two values stored here in the GUID beginning with 750, *refer to section on User Assist*

Setup.exe then creates a new process, msiexec.exe, which belongs to the Windows Installer Component. After creating this new process, setup.exe exits (Event ID 593). In the system log, two events are logged indicating that the windows installer is starting. Event ID 7035, the windows installer service was successfully sent a start control, and Event ID 7036, the windows installer service entered the running state. MSIexec.exe then creates numerous keys in the registry, of which we will only mention the most important:

$HKEY\_LOCAL\_MACHINE \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion \setminus Installer$
$\quad \setminus Folders$
$Value : c : \setminus ProgamFiles \setminus MySQL$
$HKEY\_LOCAL\_MACHINE \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion \setminus Installer$
$\quad \setminus UserData \setminus S-1-5-18 \setminus Products \setminus HEX_Value_1 \setminus InstallProperties$
$Set\ of\ values\ specifying\ the\ installation\ properties$

When the installation is completed, this event is logged into application log as event 11707. Now that the installation is completed, the service will be installed. In the security log, a new process is created which is the MySQLInstanceConfig.exe and in the registry, a new value is set:

$HKEY\_CURRENT\_USER \setminus Software \setminus Microsoft \setminus Windows \setminus ShellNoRoam \setminus MUICache$
$Value : c : \setminus ProgramFiles \setminus MySQL \setminus MySQLServer \setminus bin \setminus MySQLInstanceConfig.exe$

When the service is being installed, it sets an environment variable and its configuration in the Services key, and since the MySQL server connects to the localhost or any other host, the TCPIP settings also need to be modified. Lastly, MySQL logs to the Application event log, so that also needs to be set:

$HKEY\_LOCAL\_MACHINE \setminus System \setminus CurrentControlSet \setminus Control \setminus SessionManager$
$\quad \setminus Environment$
$HKEY\_LOCAL\_MACHINE \setminus System \setminus CurrentControlSet \setminus Services \setminus MySQL$
$HKEY\_LOCAL\_MACHINE \setminus System \setminus CurrentControlSet \setminus Services \setminus Tcpip \setminus Parameters$
$HKEY\_LOCAL\_MACHINE \setminus System \setminus CurrentControlSet \setminus Services \setminus EventLog$
$\quad \setminus Application \setminus MySQL$

Now that the service has been installed, it can be started. When it is started, a new process is created in the security log, the process is mysqld-nt.exe. Following this event are two events logged in the system log which are: event 7035, MySQL service was sent a start control, and event 7036, MySQL service entered the running state. Now the service has started and it will connect to the host, so in the security log an event indicating Firewall detected an application listening for incoming traffic (Event ID 861) is logged and the application log will record that mysqld-nt is ready for connections (Event ID 100).

6.2.8 Network

Network Configuration Information

It is always helpful to have the network configuration information handy. The Network key shown below contains several CLSID-named objects with their corresponding network configuration information. This key also includes the following subkeys: Connections, NcQueue, and SharedAccessConnection (which determines if a shared connection is enabled). The path of this Network key is:

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Control \ Network

Recent Network Settings

Imagine examining the Windows process firewall log from the image of a machine without any knowledge about the network(s) that the machine was connected to. The firewall log alone can be somewhat confusing if you are not very knowledgable in networks. But knowing which network adapters are or were installed on the machine along with their respective IP addresses and default gateway can provide a well established starting point. The registry stores recent network adapter settings, even though a network connection is disconnected, the settings are retained. This information can be found under the following key:

HKEY_LOCAL_MACHINE \ System \ CurrentControlSet \ Services \ Tcpip \ Parameters \Interfaces \ GUID

You can also know the system's IP address and default gateway information through the following key:

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Services \ adapterGUID \Parameters \ TCPIP

Wireless Network

In regards to wireless network connections, if the adapter is using Windows Wireless Zero Configuration Service, the network information will be found in the following key:

$HKEY\_LOCAL\_MACHINE \setminus SOFTWARE \setminus Microsoft \setminus WZCSVC \setminus Parameters$
$\setminus Interfaces \setminus GUID$

Mapped Network Drives

Another two keys of interest here are the Map Network Drive MRU and the MountPoints2 keys, shown below. The former provides a list of the mapped network drives along with the server name and the shared folder. The MountPoints2 key contains permanent subkeys regarding mapped network drives. These subkeys can only be removed by opening the regedit.exe and manually removing them.
The two registry keys discussed are the following:

$HKEY\_CURRENT\_USER \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion$
$\setminus Explorer \setminus MapNetworkDriveMRU$
$HKEY\_CURRENT\_USER \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion$
$\setminus Explorer \setminus MountPoints2$

6.2.9 Internet Explorer

In case the user was using Internet Explorer to browse the internet we can retrieve the browser settings, web form data, auto complete passwords and web addresses, a list of the last typed URLs, and the default download directory.

The browser settings can provide information as to what is the start page, local page directory, and search bar. Some trojan horses are placed in these locations and are executed whenever IE is executed. If any suspicious names are found, this information can be correlated with the security log or in some cases with the firewall log. For example, if we find that there is a frequent connection to some IP address in the firewall log, we check the time each time this connection is opened. If we find that it is the same time as when the Internet Explorer is started, then we assume that either the user always goes to that website, or that there is a malware. From the information collected from the Internet Explorer settings' registry key we find that the start page is the same as the mysterious recurring IP address.

The registry key where we can get the Internet Explorer setting is:

HKEY_CURRENT_USER \ Software \ Microsoft \ InternetExplorer \ Main

In regards to the web form data, auto complete passwords and auto complete web addresses, these are very case specific, so we will not go over the correlation for the time being but just point their locations:

HKEY_CURRENT_USER \ Software \ Microsoft \ ProtectedStorageSystemProvider \ SID \InternetExplorer \ InternetExplorer – q
HKEY_CURRENT_USER \ Software \ Microsoft \ ProtectedStorageSystemProvider \ SID \InternetExplorer \ InternetExplorer – URL
HKEY_CURRENT_USER \ Software \ Microsoft \ InternetExplorer \ IntelliForm
HKEY_CURRENT_USER \ Software \ Microsoft \ ProtectedStorageSystemProvider

The list of last typed URLs can be correlated with the firewall log to get the exact time of access for each URL. As explained earlier, the registry will record the last access time of the key, but if the key contains more then one value we cannot find the access time unless we correlate with the firewall log. The key that contains the last accessed URLs is:

HKEY_CURRENT_USER \ Software \ Microsoft \ InternetExplorer \ Main

The default download directory can be of use to determine a file's origin. For example, if we are monitoring some file activity in the security log, and we know that the file is in the download directory, we can assume that the user downloaded it voluntarily. Although this is not solid proof, but it adds to the facts gathered about a case. The default download directory can be found in the following key:

HKEY_CURRENT_USER \ Software \ Microsoft \ InternetExplorer

6.2.10 Most Recently Used (MRU) List and Recent Documents

Most recently used items are stored in different locations in the registry depending on their context. MRUs are of great importance for investigators and are the most suitable for correlation. In the following we list some MRUs of interest and elaborate on their usage.

HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion
\Explorer \ ComDlg32 \ OpenSaveMRU

This key lists the files that have been opened and/or saved using the Windows Explorer dialog box. Subkeys of this key are all the extension types and the corresponding files that have been opened/and or saved. Correlating these keys with the Windows security log enables us to know when these files were opened. Furthermore, correlating these two sources helps us determine any incongruence in either the registry or the log file.

HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion
\Explorer \ RunMRU

Anytime a user executes a command from the Start− > Run command line, the executed commands are stored in this key. Correlating this with the log files does not reveal much significant information, it just provides a stronger evidence to confirm the source of command execution. The security log records this information in a different way. When a command is executed, some object has to be opened, so Object Open event is logged (Event ID 592) and the object that opened this new object will be explorer.exe. But no indication that it was or was not executed through Start− >Run.

There are numerous application based MRUs, for example, if a user modified a word document and used the "SaveAs" option to save the document after modification, the name of the file is listed in the corresponding MRU. Again correlating this with the Object Open event in the security log, we can determine when the file was opened. The key that lists this type of MRU file is:

*HKEY_CURRENT_USER \ Software \ Microsoft \ Office * 10.0 *\ Common \ OpenFind*
\MicrosoftWord \ Settings \ SaveAS \ FileNameMRU

Another example of the usefulness of MRUs for correlation is the following: assume the user had VNC installed and was logging on remotely to another machine. The log files will show an instance of VNC opening, then the firewall log will indicate a connection from the machine to some address. Now the registry's VNC MRU list will indicate the name of the hosts that were connected to using the VNC viewer. This latter piece of evidence will show that VNC was actually used to connect to the host. Without this link, the evidence will be questionable. The key containing the VNC's MRU list is:

$HKEY\_CURRENT\_USER \setminus Software \setminus ORL \setminus VNCviewer \setminus MRU$

RecentDocs is a key that falls under this category, although it is not directly called MRU. This key is similar to the OpenSaveMRU key. It contains recently opened files which are categorized based on their extension. The data stored in this key is in binary format which corresponds to the filename. To further elaborate on the RecentDocs key we will give an example of opening a text document in notepad.exe. When the file is opened, the security log will record that the process notepad.exe has been created, then the RecentDocs key will be updated:

$Process\ Created(Security\ Log\ EventID\ 592):$

$\quad Process\ FileName:\ notepad.exe$

$\quad Creator\ Process:\ explorer.exe$

$Registry\ Key\ Changed:$

$\quad Key:\ HKEY\_Current\_User \setminus Microsoft \setminus Windows \setminus CurrentVersion \setminus Explorer \setminus$

$\qquad RecentDocs \setminus \#$

$\quad Value:\ name\ of\ TXT\ file$

6.2.11 User Assist

This is a key in the registry that maintains a list of objects the user has accessed. It is found in the following location:

$HKEY\_CURRENT\_USER \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion$
$\qquad \setminus Explorer \setminus UserAssist$

There are two subkeys under this key. The subkeys are in the form of GUID. The first key (beginning with "5E6") contains recent accessed objects corresponding the Internet Explorer toolbar. The second key (beginning with "750") contains recent accessed objects of the Active Desktop [5]. The data in this key is encrypted with a very basic encryption algorithm called ROT-13. The algorithm substitutes each character by another character 13 positions away.

This key can be correlated with Object Open event from the security log giving us the time the object was opened and a confirmation of the authenticity of both pieces of evidence since the occurrence in one is confirmed by the occurrence in the other.

6.2.12 Firewall Policy Change

In a number of attacks the firewall needs to be deactivated in order for the malicious activity to be successful and not be detected. When a firewall is deactivated, the firewall policy is changed to have the operational mode set to off. Such an activity is logged in the security log, and the registry key pertaining to the policy is modified. The sequence of events leading to firewall deactivation is as follows:

Process Created(Security Log EventID 592) :

  Process FileName : *c* : \windows \\ *system32* \\ *rundll32.exe*

  Creator Process : *explorer.exe*

Registry Key Changed :

  Key : *HKEY_Local_Machine* \\ *system* \\ *CurrentControlSet* \\ *SharedAccess* \\ *Parameters*\\
    StandardProfile \\ *EnableFirewall*

  Value : 0

Windows Firewall Operational Mode Changed(Security Log EventID 853) :

  Policy Origin : *LocalPolicy*

  Profile Changed : *Standard* (*or whatever the profile name is*)

  Interface : *All Interfaces*

  New Setting : *Operation mode* : *OFF*

  Old Setting : *Operation mode* : *ON*

Process Exited(Security Log EventID 593) :

  Process FileName : *c* : \windows \\ *system32* \\ *rundll32.exe*

Chapter 7

Design and Implementation

In this chapter we discuss the design and implementation of both of our proposed approaches. We use Java as our implementation programming language for the log analysis due to the choice of used APIs. This implementation is built as plug-in for the digital forensic framework developed in our Computer Security Laboratory. The functionality implemented is a proof-of-concept, thus the functionality is limited to serve only the purpose of verifying that both of our approached are successful.

7.1 Formal Log Analysis

To analyze the logs we parsed each log format as-is into a mySQL database. The choice of having the logs in a database rather than XML, for example, is due to the benefits of quick and easy queries. It is much faster querying a database rather then searching though an XML file. We limit our analysis to Windows system, security, application, and firewall logs. The system, application, and security logs have the same fields with the exception of the description field. Each of the three logs stores different types of information into the description field, and in most cases this is the most important field as it contains the details of the events. When looking for a specific entry in the description field we simply search through the text. Although a string search is an expensive operation, we first search for all the fields besides the description minimizing the number of string searches required to a sub-query.

An additional feature we added to this implementation was statistical analysis. The statistical analysis performed is basic but statistical graphs which enable an analyst to pinpoint any suspicious narrowing down a segment of the logs that needs to be analyzed. To draw the charts we used the

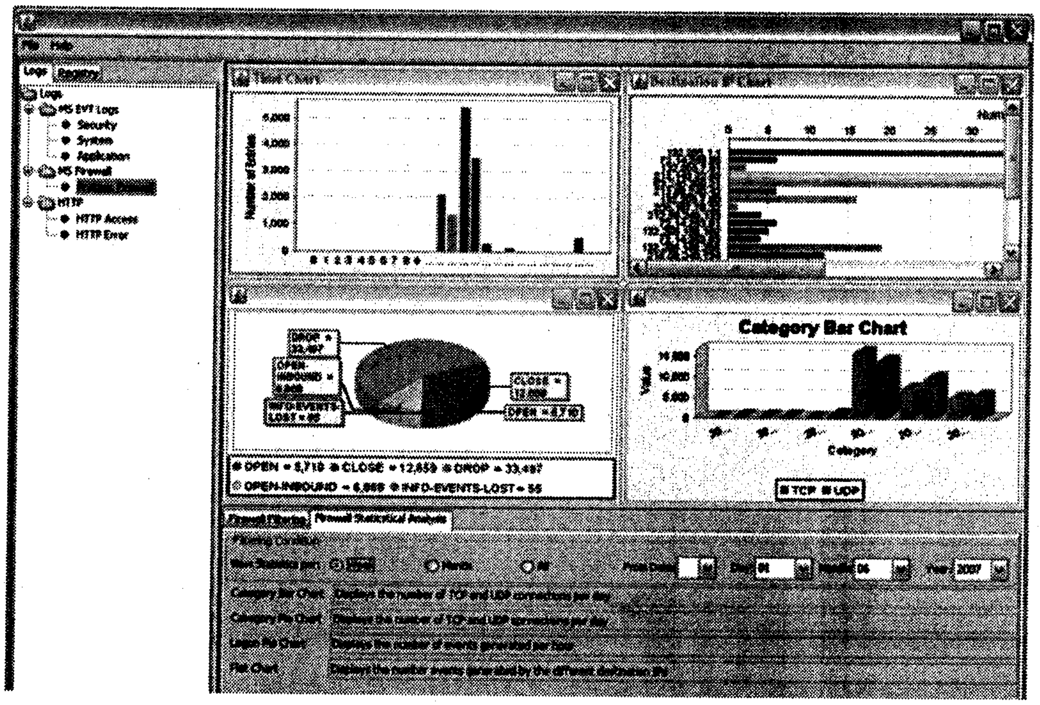
Time Chart
Category Bar Chart
Category
TCP UDP
OPEN = 5,719 CLOSE = 12,653 DROP = 33,487
OPEN-INBOUND = 6,869 INFO-EVENTS-LOST = 55

Figure 5: Statistics Charts

open source Java API JFreeChart, which provides a straightforward interface for representing the statistical analysis. An example of the statistic charts GUI is shown in figure 5 where events can be shown by required week, month, and year. For example, an investigator knows that the attack happened in a specific month, but no specific date is determined. Rather than having to analyze a month worth of events, the analyst can perform the statistics on the month in question. Based on those results, the analyst might want to look at a specific week where there seems to be abnormal activity. By looking at the charts for one week, the analyst is even able to limit the scope of the investigation to a couple of days thus reducing the analysis time significantly. We provide some visual example of these chars: Figure 6 shows us the type of actions in the events represented in a pie graph, having lots of dropped packets can allude to suspicious or in some cases reconfiguration, however, if the trends of dropped packets in a certain time period is different then the rest this increases the likelihood of suspicious activity. Figure 7 shows the number of UDP and TCP packets per day. Figure 8 shows the number of events per destination IP for the desired time period enabling us to see which IP or possibly websites are access the most. Finally Figure 9 shows the number of events per hour which is especially useful if there was a lot of activity outside the regular operational hours.

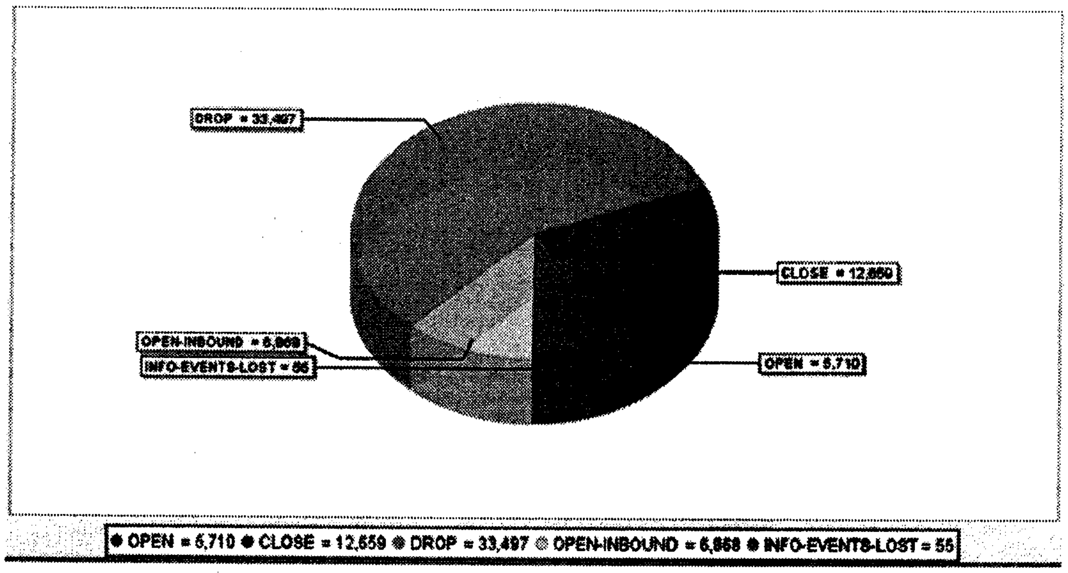

Figure 6: Actions Pie Chart

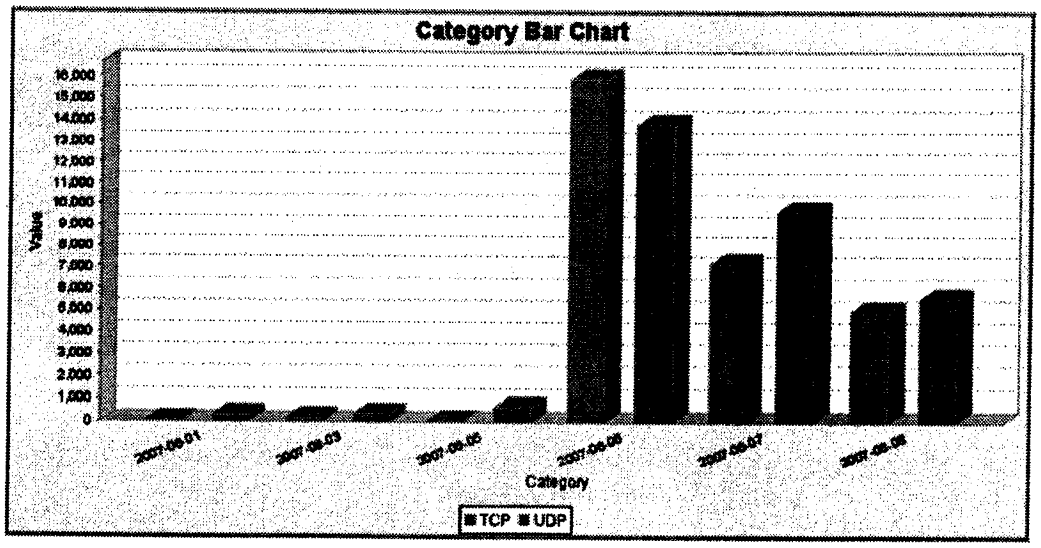

Figure 7: Type of Packets Bar Chart

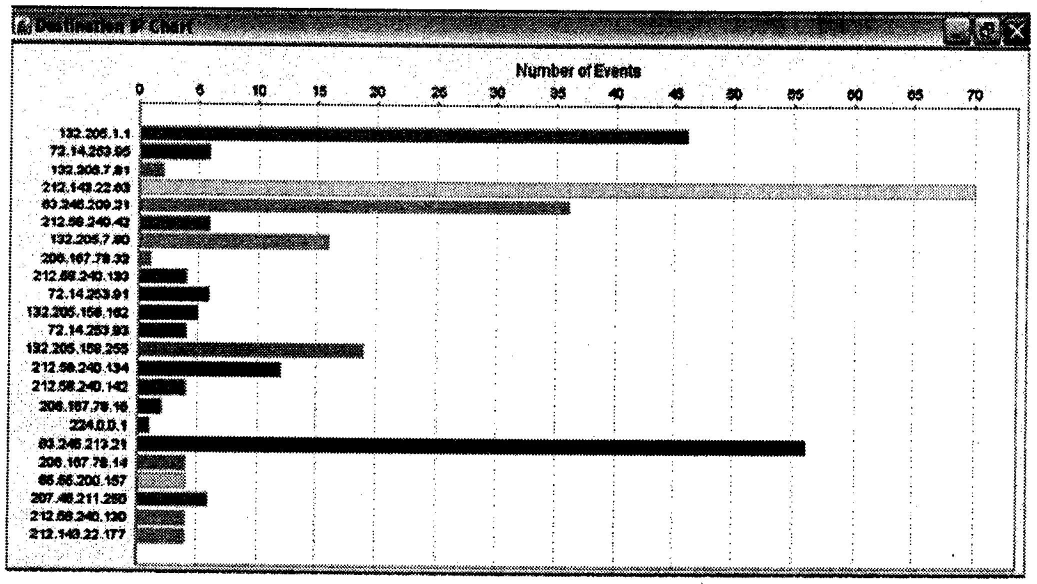

Figure 8: Number of Events per IP Bar Chart

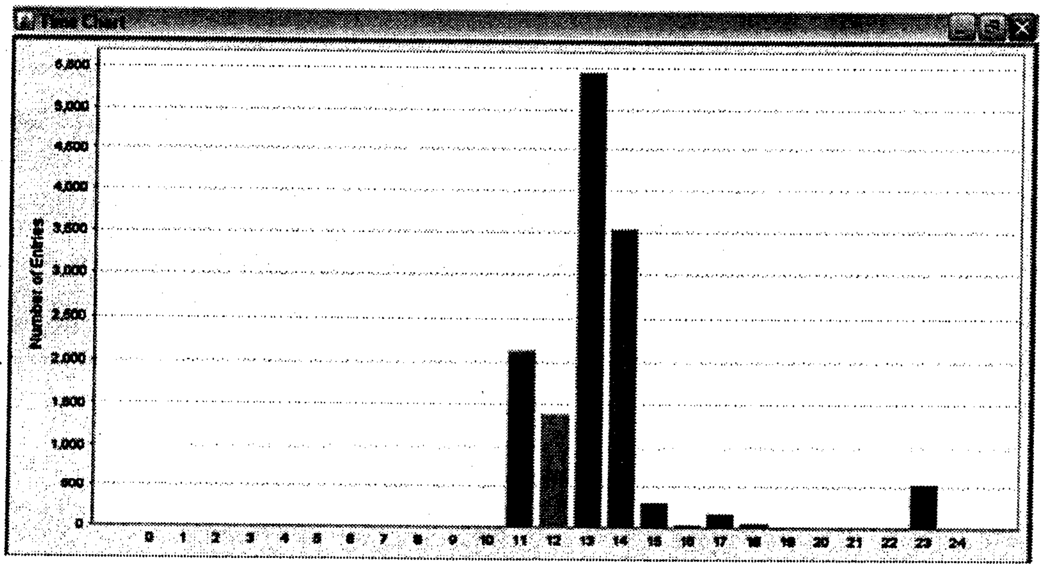

Figure 9: Number of Events per Hour Bar Chart

7.2 Methodology for Registry and Log Correlation and Analysis

In this section we present an approach for correlation which will be used to find relationships and check rules defined for different sources of evidence. If a rule or relationship is not satisfied, an alert should be triggered to inform of a possible anomaly. From hereon, rules and relationships will be called rules since a relationship can be defined in rules.

Rules can be divided into three categories: 1- Rules that are executed at initialization, these rules serve to give the analyst an overall idea of the system and configuration at hand. 2- Rules that are executed upon user request and support variables, for example the result of the first category of rules shows a suspicious service set to initialize at startup, the analyst may wish to check rules that apply to that service only. 3- Rules created by users, these rules are mainly composite rules built at runtime as the analyst goes through the analysis, she may wish to check case specific rules.

Rules are defined in XML files. To demonstrate this, we will use as an example the rule defined in Firewall Policy Change. The time attribute is used to determine the time gap, in minutes, between the events listed in the rule. In this example, the time is t_1 in all events, meaning that the events occur consecutively with less then a minute difference. Assuming that an event A occurs and than we are looking for event B with max time difference of 2 minutes, the time for A would be t_1, and the time for B would be $(t_1 + 1)$.

```
<?XMLversion = "1.0"? >
< RULE name = "Firewall Policy Change >
  < SOURCE name = "Security Log" >
    < EVENT id = "592" >
      < NAME > Process Created < \NAME >
      < FILE_NAME > c : \windows \ system32 \ rundll32.exe < \FILE_NAME >
      < CREATORPROCESS > explorer.exe < \CREATORPROCESS >
      < TIME > t1 < \TIME >
    < \EVENT >
  < \SOURCE >
  < SOURCE name = "Registry" >
      < KEY > HKEY_Local_Machine \ system \ CurrentControlSet \ SharedAccess\
                     Parameters \ StandardProfile < \KEY >
      < VALUE > EnableFirewall < \VALUE >
      < DATA > 0 < \DATA >
      < TIME > t1 < \TIME >
  < \SOURCE >
  < SOURCE name = "Security Log" >
    < EVENT id = "853" >
      < NAME > Windows Firewall Operational Mode Changed < \NAME >
      < POLICY_ORIGIN > LocalPolicy < \POLICY_ORIGIN >
      < PROFILE_CHANGED > Standard < \PROFILE_CHANGED >
      < INTERFACE > All Interfaces < \INTERFACE >
      < NEW_SETTING > Operation mode : OFF < \NEW_SETTING >
      < OLD_SETTING > Operation mode : ON < \OLD_SETTING >
      < TIME > t1 < \TIME >
    < \EVENT >
  < \SOURCE >
< \RULE >
```

After the XML file is passed to the parser, all the source elements are extracted and passed to the verifier. The verifier processes the source elements based on their name to determine what attributes to check for. The first source is checked, if it exists, we get the time and look for the second source in the given time frame. If the second source exists, we check the third and so on. If

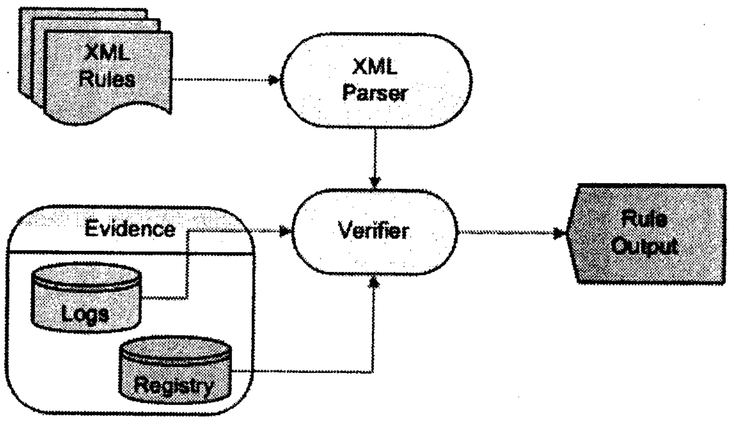

Figure 10: Correlation System

any of the sources do not exist, then the rule all together does not hold. Figure 10 shows the overall system and how the different components interact.

7.3 Registry and Log Correlation: Case Study

In this case study we show the benefits of correlating the registry with the logs as opposed to relying only on the logs for the analysis. The use of the registry should provide details not found in the logs and a means to verify the authenticity of the evidence at hand. The scenario we used for our case study is as follows:

Mr. Smith always seems to have problems with his computer and resorts to the IT administrator for assistance. The administrator was getting fed up of Mr. Smith's continuous complaints and lack of basic computer knowledge to resolve intrinsic problems. One day Mr. Smith goes to the IT administrator, as usual, requesting some work to be done on his machine. Mr. Smith was going to be out of the office for the rest of the day, so he let the administrator log on remotely and do whatever she needed to does to resolve the problem.

When the administrator logged on remotely, she saw that Mr. Smith's outlook was left opened and there was a progress report in the drafts folder addressed to the manager which has not been sent yet. The administrator decided to get back at Mr Smith for all the unnecessary hassle he causes. After finishing the work she had to do for Mr. Smith, she wrote a word document addressed to the manager, allegedly signed by Mr. Smith himself, and attached the document to the drafted email. The document was called details.doc, so it appeared as if it is the details of the progress report but contained a letter complaining about the manager's lack of competency and threatening him to go to upper management. When Mr. Smith returned, he automatically updated the outlook mail, as he normally does, and the progress report was sent to the manager. Upon receiving this email, the manager wanted to fire Mr. Smith for his outrageous behavior. However, Mr. Smith refused to be accused of such an act since he was sure he had done no such thing. The email was sent from his account, but he claims that someone had framed him. An external third party was called upon to resolve this dispute, and an investigator was assigned to examine Mr. Smith's claim.

The investigator images Mr. Smith's hard disk, getting the registry and the log files. First thing he needs to determine if Mr. Smith actually wrote that letter. Knowing that it is a word document he checks the registry key:

$$HKEY\_Current\_User \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion \setminus Explorer$$
$$\setminus RecentDocs \setminus .doc$$

This key shows the recently opened files. The investigator finds a value showing the filename details.doc. So the file was at least opened on that machine. The way this key functions is there is a value MRUListEx under the same key which determines the order in which the files were opened. The investigator looks at this value and finds that the details.doc is the second on the list. If it was first on the list, the investigator would have just looked at the last write time of the key and checked the log to determine who was logged in at that time.

Knowing that there was another word document opened after Details.doc, the investigator now has to look at the security log file to find when this file could have been last accessed. The security log indicates that there are two "process begin" events (Event ID 592) with the process name being winword.exe at different times. The event of interest is the first occurrence of the two events. However, the time of that event is the time that Mr. Smith claims to be out of the office. If that is the case, only the administrator could have logged in remotely. To test this hypothesis, the investigator checks for a remote log on rule around the time of the first occurrence of winword.exe process begin and the time that Mr. Smith left the office. Assuming the time the Mr. Smith left the office is t_1 the rule will be as follows:

$Process\ Created\ (EventID\ 592)$

$User:\ NT\ Authority \setminus System$

$Process\ name:\ c:\setminus windows \setminus system32 \setminus winlogon.exe$

$Creator\ process:\ smss.exe$

$Logon\ ID:\ x$

$Time: t_2 > t_1$

$Session\ Disconnected\ from\ Winstation\ (EventID\ 683)$

$User:\ NT\ Authority \setminus System$

$Session\ user:\ administrator$

$Logon\ ID:\ Y$

$Session\ name:\ console$

$Client\ name: --$

$Client\ address:\ --$

$Time:\ t_2 > t_1$

$Process\ Created\ (EventID\ 592)$

$User:\ NT\ Authority \setminus System$

$Process\ name: c:\setminus Program\ Files \setminus PC\ Connectivity\ Solution \setminus NclInstaller.exe$

$Creator\ process:\ serviceLayer.exe$

$Logon\ ID:\ x$

$Time:\ t_2 > t_1$

$Session\ Reconnected\ to\ Winstation\ (EventID\ 682)$

$User:\ NT\ Authority$

$Session\ user:\ administrator$

$Logon\ ID:\ Y$

$Session\ name:\ RDP\_TCP\sharp 3$

$Client\ name:\ administrator's\ machine$

$Client\ address:\ administrator's\ IP\ address$

$Time:\ t_2 > t_1$

$Process\ Exited\ (EventID\ 593)$

$User:\ NT\ Authority \setminus System$

$Process\ name:\ c:\setminus windows \setminus system32 \setminus winlogon.exe$

$Time:\ t_2 > t_1$

The time in all the events is: $t_2 > t_1$, meaning the time (in minutes) after Mr. Smith left his office. The rule verifier component searches for a process created event with process name = winlogon.exe occurring after the time Mr. Smith left his office. Winlogon is the process used for Windows logons and logoffs. When the event is found. its time is recorded and the events that follow should all be within the same minute. Normally these 5 events would occur even in the same second, but we take into consideration that Windows is a multi-threaded operating system, thus there might be other threads occurring at the same time. After the process winlogon.exe begins, it disconnects the console session so that the remote session ($RDP\_TCO\sharp3$) can be connected. The process NclInstaller.exe then is executed which is used to establish the connection for remote desktop. This is the normal execution path for a remote desktop connection. The rule continues as such:

$Process\ Created\ (EventID\ 592)$

$User:\ administrator$

$Process\ name:\ c:\backslash Program\ Files \backslash Microsoft\ Office \backslash Office11 \backslash winword.exe$

$Login\ ID:\ Y$

$Time:\ t_3$

$Registry\ Key\ Set$

$Key:\ HKEY\_Current\_User \backslash Software \backslash Microsoft \backslash Windows \backslash CurrentVersion$
$\backslash Explorer \backslash RecentDocs \backslash .Doc$

$Value:\ valueNumber$

$Data:\ Hex\ value\ of\ the\ file\ name\ (Details.doc)$

$Time:\ t_4$

$Registry\ Key\ Changed$

$Key:\ HKEY\_Current\_User \backslash Software \backslash Microsoft \backslash Windows \backslash CurrentVersion$
$\backslash Explorer \backslash RecentDocs \backslash .Doc$

$Value:\ MRUListEx$

$Time:\ t_4$

$Process\ Exited\ (EventID\ 593)$

$User:\ administrator$

$Process\ name:\ c:\backslash Program\ Files \backslash Microsoft\ Office \backslash Office11 \backslash winword.exe$

$Time:\ t_5$

$Session\ Disconnected\ from\ Winstation\ (EventID\ 683)$

$User:\ NT\ Authority \setminus System$

$Session\ user:\ administrator$

$Logon\ ID:\ Y$

$Session\ name:\ RDP\_TCP\sharp 3$

$Client\ name:\ administrator's\ machine$

$Client\ address:\ administrator's\ IP\ address$

$Time:\ t_6$

$Registry\ Key\ Changed:$

$Key:\ HKEY\_Current\_User \setminus Software \setminus Microsoft \setminus Windows \setminus CurrentVersion$

$\setminus Explorer \setminus Remote$

$Time:\ t_6$

In this part of the rule, the investigator is for a winword.exe process begin that occurred at t_3 which is the event that should have opened details.doc, thus it would have been recorded in the RecentDocs key of the registry. The last "key changed" event of the rule shown above is used to validate that the evidence has not been tampered with. When a remote desktop session is initialized, this registry key is set and system information related to this remote connection is stored there. When the session ends, the values are deleted, hence the last write time of this key should be the same time as the remote desktop session ended.

Results

The analysis of this case study was confirmed using regmon.exe from SysInternals [2]. This tool provided considerable help throughout the entire registry analysis as it shows the keys that are being accesses either for reads, writes, or deletes. We implemented the case study gathering both the logs and the registry and preformed the analysis. After getting the results, we confirmed them by redoing the scenario this time with live monitoring of the logs and registry. The comparison showed the same results we had.

7.4 Log Analysis: Case Study

To better comprehend the power of our approach, we will consider the following case: In an office environment, there are several hosts connected to a database server. An intrusion detection system is setup as a security measure for the server. The server can be accessed from outside the company so that the employees can connect to the server from client offices. Figure 7.4 depicts the scenario.

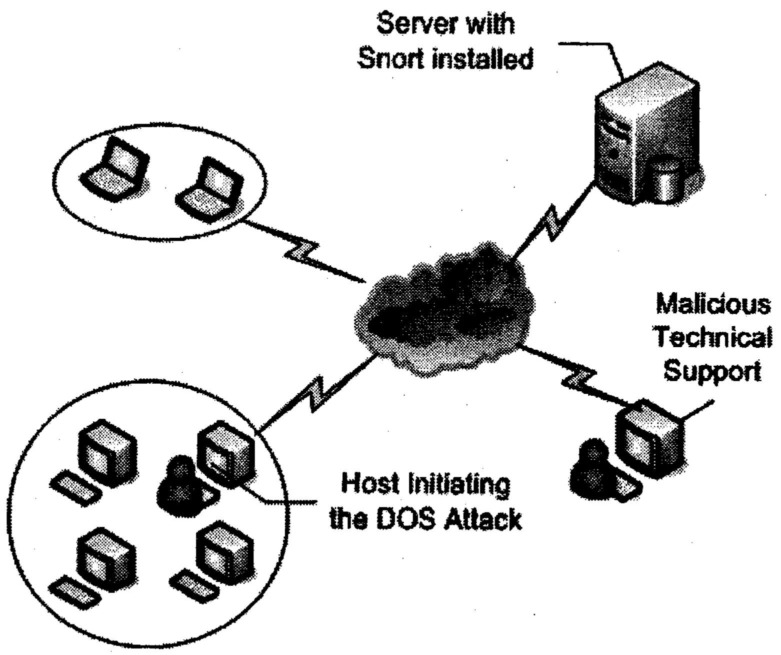

Figure 11: Case Scenario

The network administrator discovers that there was a denial of service attack on the server. The IDS (SNORT) determines a SYN attack which caused the server to halt. An investigator is called upon to investigate this incident and prosecute the liable person.

The investigator gathers the server network logs and starts searching for possible clues. Generally, when analyzing logs, an investigator will either scan through the logs manually using a simple editor which may provide some filtering capabilities, or create a script to serve his purpose. Using our methodology, the investigator only defines a trace he wants to look for, and since the logs are correlated, he doesn't even need to jump from one log file to another. In this case the system administrator has reported a SYN attack which is detected by SNORT and logged with SID = 1:526 [77]. To locate the occurrence of this attack the administrator uses the following trace from the SNORT logs:

$$\langle\langle SNORT\rangle\rangle\langle x_1.\, Attack(td_x, sID_1, sIP_x, dIP_x).x2 \nrightarrow \varepsilon\rangle \mathtt{tt}$$
$$Where\ sID_1 \;=\; 1:526$$

Once found the investigator examines the event and realizes that this IP is within the same network of the company. Knowing the date and time of the first occurrence of the attack the investigator looks for connections opened on the same date and time from that host to the server.
Until now, the events have been mostly gathered from the IDS logs. A glimpse of the advantage and ease of our system in correlation is shown here. In the following, the Windows system, application, and security logs are correlated along with the network logs. The logs being correlated into the same tree, we can define our trace and the model checker will search for it through the tree. However, to improve efficiency, we can specify which log we want to examine thus reducing the scope of our search. This can be done by adding $\langle\langle LogSource\rangle\rangle$ right before the corresponding pattern or trace. *LogSource* in this case being SNORT (like in the pattern above), SEC (security log), SYS (system log), or APP (application log). It is possible to specify more then one log e.g. $\langle\langle SYS, SEC\rangle\rangle$, or all logs excluding one specific log e.g. $\langle\langle \mathcal{L}\backslash SNORT\rangle\rangle$.
Getting back to our case, the event reflecting opening a connection can be found using:

$$\langle\langle SEC, SYS, APP\rangle\rangle\langle x_1.OpenConnection(p_x, pt_x, proc_{TCP}, uID_x, ip_1ac_x, dt_1).x_2 \looparrowright \varepsilon\rangle \mathtt{tt}$$
$$Where:\ ip_1\ is\ the\ server's\ IP\ and\ dt_1\ is\ the\ time\ from\ the\ previous\ step$$

Looking at the process that opened the connection, it seems to be some unknown executable. The investigator decides to see what process executed this executable. The first step is to find where the executable was initialized:

$$\langle\langle SEC\rangle\rangle\langle x_1.ProcessBegin(p_1, u_x, cp_x, pv_x, e_x).x_2 \looparrowright \varepsilon\rangle \mathtt{tt}$$
$$Where:\ p_1\ is\ the\ malware$$

Once found, the caller of this process can be determined from the event, which is the cp_x. Using the same trace as the above but with $p_1 = callerprocess$ the investigator can find out the name and executable file of this process. Once found, it turns out to be *BKO.exe*, which is the executable file of backOrifice2000, a backdoor. The next step is to track down the first time the back door has been executed which can be determined through the following:

$$\begin{aligned}&\langle\langle SEC\rangle\rangle\langle x_1.Logon(u_x, uID_x, g_x, d_x, p_x, lt_x).x_2.\\ &\qquad\qquad ProcessBegin(p_1, u_x, cp_x, pc_x.e_x).x_3.Logoff(u) \looparrowright x_1\rangle tt\\ &\qquad \neg\langle x_4.ProcessBegin(p_1, u_x, cp_x, pc_x.e_x).x_5 \looparrowright \varepsilon\rangle tt\\ &Where: p_1 = Bo2k.exe\end{aligned}$$

The first part of the above query looks for sub-traces containing a session, which is specified between a login and a logout, during which the back door process has been executed. The second part of the query limits the response to the first occurrence of the pattern by enforcing the constraint that no *ProcessBegin* event has happened before the selected sub-trace can contain a *ProcessBegin* event for the back door.

Since at each log on to the system, the back door program is executed, the investigator suspects that there is an entry in one of the start-up registry keys or files of the system for Bo2k.exe. Searching through these start-up places which are fortunately limited to a limited registry keys and few files, the investigator finds the entry for the back door executable in the run registry key. Therefore, it remains to track down the first session before the first execution of the back door program during which the run registry key has been modified. To do so, the investigator submits the following query:

$$\begin{array}{l}
\langle\!\langle SEC\rangle\!\rangle\langle x_1.Logon(u_x, uID_x, g_x, d_x, p_x, lt_x).x_2. \\
\qquad\qquad ProcessBegin(p_1, u_x, cp_x, pc_x.e_x).x_3.Logoff(u) \looparrowright x_1\rangle tt \\
\quad \neg\langle x_4.ProcessBegin(p_1, u_x, cp_x, pc_x.e_x).x_5 \looparrowright \varepsilon\rangle tt \\
\quad \wedge\langle x_6.logon(u_y, uID_y, g_y, d_y, p_y, lt_y).x_7.objectOpen(o_1, p_z, pn_z, tp_z, lID_z).x_8. \\
\qquad\qquad ObjectOperation(o_1, op_{SET}).x_9.logoff(u_y) \looparrowright \varepsilon > tt \\
Where : o_1 = RUN\_REG
\end{array}$$

The first two parts of the query have been explained before whereas the third part returns the session during which the malicious attacker has changed the registry key to install the back door program. Submitting the above query, the investigator can extract the user name of the *mal* user as the criminal from the substitution set.

Thus this trace shows that this user was not responsible for the attack, although she could have been easily framed. As it was shown, using our approach not only will the investigator be able to verify the admissability of his hypothesis, but she will also be capable of extracting information through the use of unbound variables in his queries. This attack scenario is depicted in figure 12

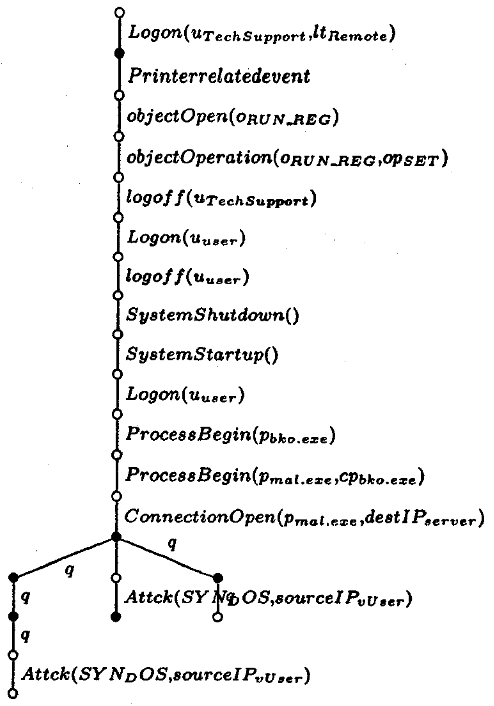
Logon(u_TechSupport, lt_Remote)
Printerrelatedevent
objectOpen(o_RUN_REG)
objectOperation(o_RUN_REG, op_SET)
logoff(u_TechSupport)
Logon(u_user)
logoff(u_user)
SystemShutdown()
SystemStartup()
Logon(u_user)
ProcessBegin(p_bko.exe)
ProcessBegin(p_mal.exe, cp_bko.exe)
ConnectionOpen(p_mal.exe, destIP_server)
q
q
q
q
Attck(SYN_DOS, sourceIP_vUser)
Attck(SYN_DOS, sourceIP_vUser)

Figure 12: Attack Scenario

Chapter 8

Conclusion and Future Work

In this thesis we proposed a model checking approach to the problem of formal analysis of logs. We model the log as a tree labeled by terms from a term algebra that represents the different actions logged by the logging system. The properties of such a log are expressed through the use of a logic that has temporal, modal, dynamic and computational characteristics. Moreover, the logic is provided by a sound and complete tableau-based proof system that is the basis for verification algorithms. The Windows logging system was studied as a use case through which we presented ideas about properties of logs such as attack traces and invariant properties. We also demonstrated how our system can be used to express signatures of malicious code and hypothesis through our case study. One question that often arises when dealing with log analysis is the integrity of the logs. In our work we demonstrate through the use of invariant properties and baselining how the integrity of the logs can be verified. A different approach can be used such as in [78] a method for making all log entries generated prior to the logging machine's compromise is described. This makes it impossible to read, modify or destroy the logs. This approach or any other similar approach for ensuring the integrity of the logs is out of our scope since we deal with logs and incidents after the fact.

We also proposed a registry correlation approach which supplements the former mentioned approach. Although the registry is used mainly to store system information, the amount of data that could be extracted from it was quite surprising. Bearing in mind that what we showed in the registry correlation chapter was what can be used for correlation with the logs, however Windows registry analysis by itself is a different issue and a different area of research. Correlation with the registry provides information that otherwise would not be available. As shown in the case study, for example, we can determine that a word document was opened, who opened it and at what time. However, without the registry it would have been impossible to know from the logs which word document was actually opened.

One of the options we considered for when implementing the log analysis was to generalize the logs. Instead of having to deal with different types of logs, we would have a generic log format and all the logs will be parsed to that format. The problem we faced was that the fields of a system log are different then the fields of a firewall log, for example. One of the solutions we thought of was to create a generic format for the different categories of log. The fields of logs within the same category can be mapped to a generic format while respecting the context of each field. Further research would be required to have the best optimized generic log format that would facilitate the analysis by eliminating the requirement of having log specific rules.

Future research efforts could also be invested in the statistical analysis we implemented. Statistical analysis can be used to complement the approaches discussed throughout this thesis. A visual representation of the logs based on statistical analysis will serve to pinpoint any suspicious activity narrowing down the analysis to a specific segment of the logs. This will minimize the time required and increase the efficiency of the analysis.

Bibliography

[1] M. Saleh and A. R. Arasteh and A. Sakha and M. Debbabi. Forensic analysis of logs: Modeling and verification. *Knowledge-Based Systems*, 20(7):671–682, October 2007.

[2] M. E. Russinovich and D. A. Solomon. *Microsoft Windows Internals.* Microsoft Press, 2004.

[3] R. F. Smith. *Windows server 2003 security log revealed.* MTG Press, 2005.

[4] Microsoft Developer Network. Windows Event Log Reference. *MSDN*, 2008. Last Visited: July 08, 2008.

[5] H. Carvey. *Windows Forensic Analysis DVD Toolkit.* Syngress Publishing, Inc., 2007.

[6] Microsoft. *Troubleshooting Windows Firewall settings in Windows XP Service Pack 2 for advanced users.* MS Help and Support, id 875357 edition, November 2007. Last accessed: July 08, 2008.

[7] Microsoft. How To Determine Audit Policies from the Registry. `http://support.microsoft.com/kb/246120`, note = "Last accessed: July 08, 2008", 2006.

[8] P. Hipson. *Mastering Windows XP Registry.* Sybex Inc, 2002.

[9] A. J. Marcella and R. S. Greenfield. *Cyber Forensics : a Field Manual for Collecting, Examining, and Preserving Evidence of Computer Crimes.* Auerbach Publications, 2001.

[10] W. Lee and S. Stolfo. Data Mining Approaches for Intrusion Detection. *In Proceedings of the 7th USENIX Security Symposium*, 1998.

[11] A. Mounji and B. L. Charlier and D. Zampunieris and N. Habra. Distributed Audit Trail Analysis. *Proceedings of the 1995 Symposium on Network and Distributed System Security - IEEE Computer Society, 102*, 1995.

[12] N. Habra and B. L. Charlier and A. Mounji and I. Mathieu. ASAX: Software Architecture and Rule-Based Language for Universal Audit Trail Analysis. *European Symposium on Research in Computer Security (ESORICS)*, 648:435–450, 1992.

[13] U. Lindqvist and P.A Porras. Detecting Computer and Network Misuse Through the Production-basedexpert System Toolset (P-BEST). *Proceedings of the 1999 IEEE Symposium on Security and Privacy, 146-161*, 1999.

[14] K. Yamanishi and J. Takeuchi and G. J. Williams and P. Milne. Online Unsupervised Outlier Detection Using Finite Mixtures With Discounting Learning Algorithms. *In Knowledge Discovery and Data Mining, 320-324*, 2000.

[15] W. Fan and M. Miller and S. J. Stolfo and W. Lee and P. K. Chan. Using Artificial Anomalies to Detect Unknown and Known Network Intrusions. *ICDM, 123-130*, 2001.

[16] H. Carvey. KDD-99. *The Fifth International Conference on Knowledge Discovery and Data Mining*, 1999. Last accessed: July 08, 2008.

[17] P. Ma. Log Analysis-Based Intrusion Detection via Unsupervised Learning. Master's thesis, School of Informatics, University of Edinburgh, 2003.

[18] W. Lee and S. J. Stolfo. A Framework for Constructing Features and Models for Intrusion Detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):227–261, November 2000.

[19] J. Ryan and M.J. Lin and R. Miikkulainen. Intrusion Detection with Neural Networks. *Advances in Neural Information Processing Systems, 943 - 949*, 1998.

[20] M. N. A. Khan and I. Wakeman. Machine Learning for Post-Event Timeline Reconstruction. *The 7th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting*, 2006.

[21] H. Gunes Kayacik and A. Nur Zincir-Heywood and Malcolm I. Heywood. Multiple Self-Organizing Maps for Intrusion Detection. *Engineering Applications of Artificial Intelligence*, 20(4):439–451, 2007.

[22] W. Ren and H. Jin. Distributed Agent-Based Real Time Network Intrusion Forensics System Architecture Design. *AINA, Proceedings of the 19th International Conference on Advanced Information Networking and Applications*, 1(177-182), 2005.

[23] T. Y. Lin. Anomaly Detection: A Soft Computing Approach. *Proceedings of the 1994 Workshop on New Security Paradigms, 44-53*, 1994.

[24] S. T. Echmann and V. Giovanni and R. A. Kemmerer. STATL: An Attack Language for State-Based Intrusion Detection. *ACM Workshop on Instrusion Detection Systems*, 10(1-2):71 – 103, 2002.

[25] A. P. Kosoresow and S. A. Hofmeyr. Intrusion Detection via System Call Traces. *IEEE Computer Society Press*, 14(5):35–42, 1997.

[26] T. Lane and C. Brodley. Temporal Sequence Learning and Data Reduction for Anomaly Detection. *ACM Transactions on Information and System Security*, 2:295 – 331, 1999.

[27] P. Helman and G. E. Liepins. Statistical Foundations of Audit Trail Analysis for the Detection of Computer Misuse. *IEEE Transactions on Software Engineering*, 19(9):886–901, 1993.

[28] H. S. Javitz and A. Valdes. The SRI IDES Statistical Anomaly Detector. *Proceedings of the IEEE Symposium on Research in Security and Privacy, 316-326*, 1991.

[29] H. S. Vaccaro and G. E. Liepins. Detection of Anomalous Computer Session Activity. *Proceedings of the IEEE Symposium on Research in Security and Privacy, 280-289*, 1989.

[30] L. Lankewicz and M. Benard. Real-time Anomaly detection Using a Nonparametric Pattern Recognition Approach. *Proceedings of the of 7th Computer Security Applications Conference*, 85:160–165, 1991.

[31] D. Endler. Intrusion Detection: Applying Machine Learning to Solaris Audit Data. *Proceedings of the Computer Security Applications Conference, 80-89*, 1998.

[32] D. Xu and P. Ning. Alert Correlation through Triggering Events and Common Resources. *Proceedings of the 20th Annual Computer Security Applications Conference - ACSAC 04, 360-369*, 2004.

[33] F. Cuppens. Managing Alerts in a Multi-intrusion Detection Environment. volume 10 of *14*, pages 22–31. In Proceedings of the 17th Annual Computer Security Applications Conference, December 2001.

[34] K. Julisch. Clustering Intrusion Detection Alarms to Support Root Cause Analysis. volume 6 of *4*, pages 443–471. ACM Transactions on Information and System Security, Nov 2003.

[35] S. Staniford, J. Hoagland, and J. McAlerney. Practical Automated Detection of Stealthy Portscans. Journal of Computer Security, 10(1/2):105136., December 2002.

[36] A. Valdes and K. Skinner. Probabilistic Alert Correlation. volume 2212, pages 54–68. In Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001), LNCS, 2001.

[37] H. Debar and A. Wespi. Aggregation and Correlation of Intrusion-detection Alerts. In Recent Advances in Intrusion Detection, LNCS, 2001.

[38] B. Morin and H. Debar. Correlation of Intrusion Symptoms: an Application of Chronicles. In Proceedings of the 6th International Conference on Recent Advances in Intrusion Detection (RAID03), September 2003.

[39] F. Cuppens and A. Miege. Alert Correlation in a Cooperative Intrusion Detection Framework. In Proceedings of the 2002 IEEE Symposium on Security and Privacy, May 2003.

[40] P. Ning, Y. Cui, and D. S. Reeves. Constructing Attack Scenarios Through Correlation of Intrusion Alerts. page 245254. In Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington, D.C., November 2002.

[41] S. Templeton and K. Levitt. A Requires/Provides Model for Computer Attacks. pages 31–38. In Proceedings of New Security Paradigms Workshop, September 2000.

[42] B. Morin, L. Me, H. Debar, and M. Ducasse. M2D2: A Formal Data Model for IDS Alert Correlation. volume 2516, pages 115–137. In Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002), 2002.

[43] P. Porras, M. Fong, and A. Valdes. A Mission-impact-based Approach to INFOSEC Alarm Correlation. volume 2516, pages 95–114. In Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002), 2002.

[44] V. Yegneswaran, P. Barford, and S. Jha. Global Intrusion Detection in the Domino Overlay System. volume 30 of *3*, pages 877–899. In Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS04), Feburary 2004.

[45] P. Stephenson. Modeling of Post-incident Root Cause Analysis. *International Journal of Digital Evidence*, 2(2):43–54, 2003.

[46] P. Stephenson. Introduction to the Digital Investigation Process Language. *CSI 30th Annual Conference*, 2003.

[47] P. Lory. A Coloured Petri Net Trust Model. *14th International Workshop on Database and Expert Systems Applications, 415-419*, 2003.

[48] C. Hosmer. Time Lining Computer Evidence, 1998. Last accessed: July 08, 2008.

[49] T. Stallard and K. Levitt. Automated Analysis for Digital Forensic Science: Semantic Integrity Checking. In *19th Annual Computer Security Applications Conference*, pages 160–167, Las Vegas, NV, USA, December 2003.

[50] K. Monroe and D. Bailey. System Base-lining: A forensic perspective, 2003. Last accessed: July 08, 2008.

[51] P. Gladyshev and A. Patel. Finite State Machine Approach to Digital Event Reconstruction. *Digital Investigation Journal*, 1(2), 2004.

[52] P. Gladyshev and A. Patel. Formalising Event Time Bounding in Digital Investigations. *Digital Investigation Journal*, 4(2), 2005.

[53] R. Leigland and A. W. Krings. A Formalization of Digital Forensics. *Digital Investigation Journal*, 3(2), 2004.

[54] C. Peikari and A. Chuvakin. *Security Warrior.* O'Reilly, 2004.

[55] W. Kruse and J. Heiser. *Computer Forensics: Incident Response Essentials.* Addison-Wesley, Boston, MA, 2002. Last accessed: July 08, 2008.

[56] M. Roger and J. Goubault-Larrecq. Log Auditing through Model-Checking. *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW'01), 220-234*, June 2001.

[57] Z. Manna and A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems. *Springer,427*, 1991.

[58] P. Naldurg and K. Sen and P. Thati. A Temporal Logic Based Framework for Intrusion Detection. *Formal Techniques for Networked and Distributed Systems FORTE 2004*, 3235:359–376, 2004.

[59] E. Nowicka and M. Zawada. Modeling Temporal Properties of Multi-event Attack Signatures in Interval Temporal Logic. *In the Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2006), 89-93*, ISBN 3-937201-02-5, September 2006.

[60] D. Ogle and H. Kreger and A. Salahshour and J. Cornprost and E. Labadie and M. Chessell and B. Horn and J. Greken and J. Schoech and M. Wamboldt. Canonial Situation Data Format: The Common Base Event V1.0.0. *International Busines Machine Corporation*, 2004.

[61] R. A. Magalhaes. Understanding Windows Logging. *Windows Security*, OS Security, 2004.

[62] Herv Schauer Consultants. Windows log files. Online Document, 2005. Last accessed: July 08, 2008.

[63] IBM. Windows Cluster Service. Last accessed: July 08, 2008.

[64] Microsoft. Microsoft - Help and Support. Online Help Website. Last accessed: July 08, 2008.

[65] Microsoft. Services and Service Accounts Security Planning Guide. *Microsoft Technet*, 2005.

[66] K. Adi, M. Debbabi, and M. Mejri. A New Logic for Electronic Commerce Protocols. *International Journal of Theoretical Computer Science, TCS*, 291(3):223–283, 2003.

[67] R. Cleaveland. Tableau-based Model Checking in the Propositional Mu-calculus. *Acta Informatica*, 27(8):725–748, 1990.

[68] K. Ivens. *Admin911: Windows 2000 Registry*. Osborne/McGraw-Hill, 2001.

[69] M. Geiger. Counter-Forensic Tools: Analysis and Data Recovery. *18th Annual FIRST Conference*, 2006.

[70] H. Carvey. The Windows Registry as a forensic Resource. *Digital Investigation, 201-205*, 2005.

[71] V. Mee and T. Tryfonasa and I. Sutherland. The Windows Registry as a Forensic Artefact: Illustrating Evidence Collection for Internet Usage. *Digital Investigation*, 3(3):166–173, 2006.

[72] F. Apap and A. Honig and S. Hershkop and E. Eskin and S. Stolfo. Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses. *Recent Advances in Intrusion Detection : 5th International Symposium, RAID 2002, Zurich, Switzerland*, , 2516:36–53, October 2002.

[73] S. J. Stolfo and F. Apap and E. Eskin and K. Heller and S. Hershkop and A. Honig and K. Svore. A Comparative Evaluation of Two Algorithms for Windows Registry Anomaly Detection. *Journal of Computer Security*, 13(4):659–693, 2005.

[74] K. A Heller and K. M Svore and A. D. Keromytis and S. J. Stolfo. One Class Support Vector Machines for Detecting Anomalous Window Registry Accesses. *3rd IEEE Conference Data Mining Workshop on Data Mining for Computer Security, 458-461*, 2003.

[75] Microsoft. CurrentControlSet\Services Subkey Entries. `http://support.microsoft.com/kb/103000`, 2006. Last accessed: July 08, 2008.

[76] Microsoft. Microsoft Security Bulletin MS02-064. `http://www.microsoft.com/technet/security/Bulletin/MS02-064.mspx`, note = "Last accessed: July 08, 2008", 2003.

[77] Snort - sourcefire inc. http://www.snort.org. Last accessed: July 08, 2008.

[78] B. Schneier and J. Kelsey. Secure Audit Logs to Support Computer Forensics. *ACM Transactions on Information and System Security (TISSEC)*, 2(2):159–176, May 1999.

Appendix A

Proofs

A.1 Proof of Finiteness

We will need the following definitions:

Definition A.1.1 Closure. *The closure $CL(\varphi)$ of a formula φ is a set of formulas that is defined as:*

$CL(Z) = \{Z\}$

$CL(\neg\varphi) = \{\neg\varphi\} \cup CL(\varphi)$

$CL(\varphi_1 \wedge \varphi_2) = \{\varphi_1 \wedge \varphi_2\} \cup CL(\varphi_1) \cup CL(\varphi_2)$

$CL([r_1 \looparrowright r_2]\varphi = \{[r_1 \looparrowright r_2]\varphi\} \cup CL(\varphi)$

$CL(\nu Z.\varphi) = \{\nu Z.\varphi\} \cup CL(\varphi[\nu Z.\varphi/Z])$

Definition A.1.2 Size. *The size $|\varphi|$ of a formula is a positive integer:*

$|Z| = 1$

$|\neg\varphi| = 1 + |\varphi|$

$|\varphi_1 \wedge \varphi_2| = 1 + |\varphi_1| + |\varphi_2|$

$|[r_1 \looparrowright r_2]\varphi| = 1 + |\varphi|$

$|\nu Z.\varphi| = 1 + |\varphi|$

Definition A.1.3 H-Ordering. *Between hypothesis sets, we define the relation $\sqsubseteq_\varphi$, relative to a formula φ, where $\mathcal{H}$ is the set of all hypothesis sets:*

$\sqsubseteq_Z = \mathcal{H} \times \mathcal{H}$

$\sqsubseteq_{\neg\varphi'} = \sqsubseteq_{\varphi'}$

$\sqsubseteq_{\varphi_1 \wedge \varphi_2} = \sqsubseteq_{\varphi_1} \cap \sqsubseteq_{\varphi_2}$

$\sqsubseteq_{[r_1 \hookleftarrow r_2]\varphi'} = \sqsubseteq_{\varphi'}$

$\sqsubseteq_{\nu Z.\varphi'} = \{(H_1, H_2) \in \sqsubseteq_{\varphi'} \mid (H_2, H_1) \in \sqsubseteq_{\varphi'} \Rightarrow H_1 \upharpoonright \nu Z.\varphi' \subseteq H_2 \upharpoonright \nu Z.\varphi'\}$

$H_1 =_{\varphi} H_2 \Leftrightarrow H_1 \sqsubseteq_{\varphi} H_2 \wedge H_2 \sqsubseteq_{\varphi} H_1$

$H_1 \sqsubset_{\varphi} H_2 \Leftrightarrow H_1 \sqsubseteq_{\varphi} H_2 \wedge H_1 \neq_{\varphi} H_2$

$\forall \varphi' \in CL(\varphi) \centerdot H_1 \sqsubseteq_{\varphi'} H_2 \Rightarrow H_1 \trianglelefteq H_2$

$H_1 \equiv_{\varphi} H_2 \Leftrightarrow H_1 \trianglelefteq H_2 \wedge H_2 \trianglelefteq H_1$

$H_1 \lhd H_2 \Leftrightarrow H_1 \trianglelefteq H_2 \wedge H_2 \ntrianglelefteq H_1$

From the definitions above, it is clear that $\forall \varphi, \varphi' \centerdot \varphi' \prec_I \varphi \Rightarrow \sqsubseteq_{\varphi} \subseteq \sqsubseteq_{\varphi'}$. The following results about $\sqsubseteq_{\varphi}$ are proved in [67]:

- $\sqsubseteq_{\varphi}$ is reflexive and transitive, i.e., a preorder.
- $H \sqsubset_{\nu Z.\varphi} H' \cup \{\sigma : \nu Z.\varphi\}$, where $H' = H \setminus \{\sigma' : \Gamma \mid \nu Z.\varphi \prec \Gamma\}$
- $\sqsubset_{\varphi}$ has no infinite ascending chains.

Definition A.1.4 Sequent ordering *For any two sequents $\tau_1 = H_1, b_1 \vdash \sigma_1 \varphi_1$ and $\tau_2 = H_2, b_2 \vdash \sigma_2 \varphi_2$, such that $\varphi_2 \in CL(varphi_1)$, the relation $\tau_1 < \tau_2$ holds, whenever one of the following holds:*
- $H_1 \lhd_{\varphi_2} H_2$
- $H_1 \equiv_{\varphi_2} H_2 \wedge |\varphi_2| < |\varphi_1|$

Lemma A.1.1 *The sequent ordering relation $<$ has no infinite ascending chain.*

Proof. Suppose we start by $\tau_0 = H_0, b_0 \vdash \sigma \in \varphi_0$, the chain $\tau_0 < \tau_1 < tau_2 \ldots$ cannot ascend infinitely, the proof follows form the facts that $|CL(\varphi_0)|$ is finite, $\lhd_{\varphi_0}$ has no infinite ascending chains, and $\forall \varphi' \in CL(varphi) \centerdot |\varphi'| < |\varphi|$. □

Theorem A.1.1 Finiteness. *For any sequent $\tau = H, b \vdash \sigma \in \varphi$ there exists a finite number of finite tableaux.*

Proof. The proof is by well-founded induction on the inverse of the sequent ordering relation, since $<$ has no infinite ascending chain, then $<^{-1}$ has no infinite descending chain.
Induction hypothesis: for all τ' such that $\tau < \tau'$, τ' has a finite number of finite tableaux.
Required: for any τ, the applicable rule of the tableau will produce a finite number of τ', where $\tau < \tau'$.

We consider here two cases: $\tau = H, b \vdash \sigma \in [r_1 \looparrowright r_2]\varphi$ and $\tau = H, b \vdash \sigma \in \nu Z.\varphi$ since the other cases are straightforward.

Case 1: $\tau = H, b \vdash \sigma \in [r_1 \looparrowright r_2]\varphi$

The only applicable rule is $R_{[]}$, it generates sequents of the form $\tau_i = H_i, b_i \vdash \sigma_i \in \varphi$, where $i \in \mathbb{I} \subseteq \mathbb{N}$, $\mathbb{N}$ is the set of natural numbers. For all i $H_i = H$, moreover $\varphi \in CL([r_1 \looparrowright r_2]\varphi)$ and $|\varphi| < |[r_1 \looparrowright r_2]\varphi|$, hence $\tau < \tau_i$. The set $\mathbb{I}$ is determined by the set $\Theta = \{\theta_i \mid \textbf{match}\ (r_1, \sigma, \theta_i) = \texttt{tt}\}$, where $|\mathbb{I}| = |\Theta|$. Therefore we need to prove that Θ is finite, which follows from the fact that σ is finite since we are dealing with finite models. So we can write $i \in \{0, 1, \ldots n\}$. The other condition of the rule is that $b_1 \times b_2 \ldots \times b_n = b$, the number of combinations if finite ($2^{|\mathbb{I}|}$) and each of these combinations produce a different tableau.

Case 2: $\tau = H, b \vdash \nu Z.\varphi$

The only applicable rule is R_ν, which produces only one sequent $\tau' = H', b \vdash \varphi[\nu Z.\varphi/Z]$, using the results about $\sqsubseteq_\varphi$ above and the definition of the sequent order relation, it is easy to prove that $\tau < \tau'$. □

A.2 Proof of Soundness

To prove the soundness, we have to prove that all successful leaves are semantically sound and that all rules of the tableau reserve soundness. We consider two cases here and the rest of the cases can be easily proved.

Theorem A.2.1 Soundness. *For any sequent $H, b \vdash \sigma \in \varphi$ with a successful tableau, $\sigma \in [\![\varphi]\!]_e^{L,H}$*

Proof. We consider the two following cases of successful leaves and rules:

Case 1: The sequent $H, \epsilon \vdash \sigma \in [r_1 \looparrowright r_2]\varphi$, is a successful leaf when $\forall\theta.\ \textbf{match}\ (\sigma, r_1, \theta)\} = \texttt{ff}$. This agrees with the semantics $[\![[r_1 \looparrowright r_2]\varphi]\!]_e^{L,H} = \{\sigma \in L \mid \forall\theta.\textbf{match}(\sigma, r_1, \theta) \Rightarrow \sigma' \in [\![\varphi]\!]_e^{L,H}\}$, since the implication ($\Rightarrow$) will evaluate to $\texttt{tt}$. Moreover, the rule $R_{[]}$ reserves soundness since it is just an expression of the semantics of $\forall$ from first order logic.

Case 2: The sequent $H, \epsilon \vdash \sigma \in \nu Z.\varphi$ is a successful leaf when $\sigma : \nu Z.\varphi \in H$. We recall the definition of R_ν and the relativized semantics of $\nu Z.\varphi$:

$$\frac{H, b \vdash \sigma \in \nu Z.\varphi}{H' \cup \{\sigma : \nu Z.\varphi\}, b \vdash \sigma \in \varphi[\nu Z.\varphi/Z]} \qquad \sigma : \nu Z.\varphi \notin H$$

$$[\![\nu Z.\varphi]\!]_e^{L,H} = (\nu [\![\varphi]\!]_{e[Z \mapsto S \cup S']}^{L,H}) \cup S'$$

where, $H' = H \setminus \{\sigma' : \Gamma \mid \nu Z.\varphi \prec \Gamma\}$ and $S' = H \restriction \nu Z.\varphi$. So, what we would like to prove is that $[\![\varphi[\nu Z.\varphi/Z]]\!]_e^{L,H'\cup\{\sigma:\nu Z.\varphi\}} = (\nu[\![\varphi]\!]_{e[Z\mapsto S\cup\{\sigma\}]}^{L,H}) \cup \{\sigma\}$. The proof relies on lemma 5.3.1 and on the properties of fixpoints. It is detailed in [67].

A.3 Proof of completeness

The proof depends on the following theorem:

Theorem A.3.1 *The sequent $\tau = H, b \vdash \sigma \in \varphi$ has a successful tableau if and only if $\tau' = H, b \vdash \sigma \in \neg\varphi$ has no successful tableau*

Proof.

Step 1: $\Rightarrow$

Suppose that both τ and τ' have successful tableaux, then by the soundness theorem $\sigma \in [\![b\varphi]\!]_e^{L,H}$ *and* $\sigma \in [\![b\neg\varphi]\!]_e^{L,H}$, which implies $\sigma \in [\![\neg b\varphi]\!]_e^{L,H}$. From the definition of the semantics, we will have $\sigma \in [\![b\varphi]\!]_e^{L,H}$ and $\sigma \notin [\![b\varphi]\!]_e^{L,H}$, which is a contradiction.

Step 2: $\Leftarrow$

We would like to prove that if τ has no successful tableau then τ' has a successful tableau. We do this by induction on the height of proof tree, starting from the leaves, i.e., prove that unsuccessful leaves imply $\sigma \in \neg\varphi$ and whenever a node in the proof tree implies $\sigma \in \neg\varphi$ its parent implies the same thing. We consider here the case for the unsuccessful leaf $H, \neg \vdash \sigma \in [r_1 \looparrowright r_2]\varphi$ and $\{\sigma \in L \mid \exists\theta \textbf{. match } (\sigma, r_1, \theta)\} = \emptyset$ and the rule $R_{[]}$. The rest of the cases match those proved in [66]. By definition, we have $H, \neg \vdash \sigma \in [r_1 \looparrowright r_2]\varphi \Rightarrow H, \epsilon \vdash \sigma \in \neg[r_1 \looparrowright r_2]\varphi$ which is a successful leaf to prove that $\sigma \in \neg[r_1 \looparrowright r_2]\varphi$. Now for the rule $R_{[]}$:

$$\frac{H, b \vdash \sigma \in [r_1 \looparrowright r_2]\varphi}{\xi_1 \quad \xi_2 \quad \ldots \xi_n}$$

By the rule $R_{[]}$, if any $\xi_i = H, b_i \vdash \sigma' \in \varphi$ does not have a successful tableau, then $\tau = H, b \vdash \sigma \in [r_1 \looparrowright r_2]\varphi$ does not have a successful tableau. In this case, by induction hypothesis, $\xi_i' = H, b_i \vdash \sigma' \in \neg\varphi$ has a successful tableau. By definition, we have $H, \neg b_i \vdash \sigma' \in \varphi$ has a successful tableau. But $b = b_1 \times b_2 \ldots \times b_i \ldots \times b_n$, so the term $b_1 \times b_2 \ldots \times \neg b_i \ldots b_n$ will equal $\neg b$ (by definition of the $\times$ operation), which means that $H, \neg b \vdash \sigma \in [r_1 \looparrowright r_2]\varphi$, will have a successful tableau, or in other words $H, b \vdash \sigma \in \neg[r_1 \looparrowright r_2]\varphi$ will have a successful tableau. □

Theorem A.3.2 Completeness. *If for a sequence $\sigma \in L$, $\sigma \in [\![\varphi]\!]_e^{L,H}$, then the sequent $H, b \vdash$*

$\sigma \in \varphi$ has a successful tableau.

Proof. The proof follows from theorems A.2.1 and A.3.1 by contradiction. $\square$

Appendix B

Log Models

B.1 Start-up

First the LSASRV.dll authentication package is loaded. NETWORK SERVICE account logs on and is assigned the required privileges.

A set of authentication packages are loaded starting with Kerberos.dll and MSV1_0.dll, followed by schannel.dll which is a TCP/SSL Security Provider Library, wdigest.dll which is the Digest Authentication Protocol library and is used for HTTP and SASL (Simple Authentication Security Layer), and finally setuid.dll Winlogon and Winlogon\MsGina register with the Local Security Authority KSecDD, Kernel Security Support Provider Interface, also registers with the LSA SAM loads a notification package scecli (Security Configuration Editor Client Engine).

After logon is initialized, the services begin to load and an instance of svchost.exe is created by services.exe and run under services.exe privilege (by being passed a token from services.exe). Then, depending on the type of service and the privileges that it should run under, we should see an account logon followed by the service created. The accounts that logon are local service and network service.

The process is executed by the following set of functions:

AuthenticationPackageLoaded(NT Authority \ System, $sess_1$*, C : \windows \ system32\ LSASRV.dll : Negotiate)*

SuccessfulLogon(NT Authority \ Network Service, $sess_2$*, advapi, 5)*

PrivilegeAssigned(NT Authority \ Network Service, {SeAuditPrivilege, SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege})

AuthenticationPackageLoaded(NT Authority \ System, $sess_1$*, c : \windows \ system32\ kerberos.dll : Kerberos)*

AuthenticationPackageLoaded(NT Authority \ System, $sess_1$*, c : \windows \ system32\ msv1_0.dll : NTLM)*

AuthenticationPackageLoaded(NT Authority \ System, $sess_1$*, c : \windows \ system32\ schannel.dll : MicrosoftUnifiedSecurityProtocolProvider)*

AuthenticationPackageLoaded(NT Authority \ System, $sess_1$*, c : \windows \ system32\ wdigest.dll : WDigest)*

AuthenticationPackageLoaded(NT Authority \ System, $sess_1$*, c : \windows \ system32\ msv1_0.dll : MICROSOFT_AUTHENTICATION_PACKAGE_V1_0)*

AuthenticationPackageLoaded(NT Authority \ System, $sess_1$*, c : \windows \ system32\ setuid.dll : Setuid)*

PrivilegeServiceCalled(NT Authority \ System, $sess_1$*, NT Local Security Authority, LsaRegisterLogonProcess(), SeTcbPrivilege, s)*

TrustedLogonProcess(NT Authority \ System, $sess_1$*, Winlogon)*

TrustedLogonProcess(NT Authority \ System, $sess_1$*, Winlogon \ MSGina)*

PrivilegeServiceCalled(NTAuthority \ System, $sess_1$*, NT Local Security Authority, LsaRegisterLogonProcess(), SeTcbPrivilege, s)*

TrustedLogonProcess(NT Authority \ System, $sess_1$*, KSecDD)*

NotificationPackageLoaded(NT Authority \ System, $sess_1$*, scecli)*

− − this package will be notified of any password changes

PrivilegeServiceCalled(NT Authority \ System, $sess_1$*, NT Local Security Authority, LsaRegisterLogonProcess(), SeTcbPrivilege, s)*

Services start to load

$ProcessCreated(NT\ Authority \setminus System, sess_1, c:\setminus windows \setminus system32\setminus$
$\quad svchost.exe, pid_z, c:\setminus windows \setminus system32 \setminus services.exe)$

$TrustedLogonProcess(NT\ Authority \setminus System, sess_1, DCOMSCM)$

$PrivilegeServiceCalled(NT\ Authority \setminus System, sess_1, LsaRegisterLogonProcess(), SeTcbPrivilege, s)$

$ProcessCreated(NT\ Authority \setminus System, sess_1, c:\setminus windows \setminus system32 \setminus svchost.exe, pid_2,$
$\quad c:\setminus windows \setminus system32 \setminus services.exe)$

$ProcessTokenAssignment(NT\ Authority \setminus System, sess_1, c:\setminus windows \setminus system32 \setminus services.exe,$
$\quad c:\setminus windows \setminus system32 \setminus svchost.exe, pid_2, s)$

$ObjectOpen(NT\ Authority \setminus Network\ Services, sess_2, SC\_ManagerObject, ServicesActive,$
$\quad hid_x, pid_2,$ "$READ\_CONTROL\ Connect\ to\ service\ controller\ Lock$
$\quad service\ database\ for\ exclusive\ access$"$, s)$

$SuccessfulLogon(NT\ Authority \setminus Local\ Service, sess_3, Advapi, 5)$

$PrivilegeAssigned(NT\ Authority \setminus Local\ Service, \{SeAuditPrivilege,$
$\quad SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege\})$

$ProcessCreated(NT\ Authority \setminus System, sess_1, serviceName, pid_x, c:\setminus windows \setminus system32\setminus$
$\quad services.exe)$

$ProcessTokenAssignement(NT\ Authority \setminus System, sess_1, c:\setminus windows \setminus system32\setminus$
$\quad services.exe, serviceName, pid_x, s)$

$SuccessfulLogon(NT\ AUTHORITY \setminus Network\ Service, sess_4, Advapi, 5)$

$PrivilegeAssigned(NT\ Authority \setminus NetworkService, \{SeAuditPrivilege,$
$\quad SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege\})$

$ProcessCreated(NT\ Authority \setminus System, sess_1, serviceName, pid_y, c:\setminus windows \setminus system32\setminus$
$\quad services.exe)$

$ProcessTokenAssignement(NT\ Authority \setminus System, sess_1, c:\setminus windows \setminus system32\setminus$
$\quad services.exe, serviceName, pid_y, s)$

The following set of functions are dependant on the configurations, services and applications are loaded along with their required libraries and other file dependencies. A list of what should be loaded can be maintained from the registry. Now the trace will be modeled as such:

Using the formal framework, this is how the process will be expressed:

$$\begin{aligned}
&\langle\!\langle SEC\rangle\!\rangle\langle x_1.\mathit{AuthenticationPackageLoaded}(u_1, sess_1, dll_{LSARV}).x_2.\\
&\quad \mathit{SucessfulLogon}(u_2, sess_2, lp_{ADV}, 5).\mathit{PrivilegeAssigned}(u_2, axs_{srvc}).\\
&\quad x_3.\mathit{AuthenticationPackageLoaded}(u_1, sess_1, dll_{kerberos}).x_4.\\
&\quad \mathit{AuthenticationPackageLoaded}(u_1, sess_1, dll_{NTLM}).x_5.\\
&\quad \mathit{AuthenticationPackageLoaded}(u_1, sess_1, dll_{schannel}).x_6.\\
&\quad \mathit{AuthenticationPackageLoaded}(u_1, sess_1, dll_{xdigest}).x_7.\\
&\quad \mathit{AuthenticationPackageLoaded}(u_1, sess_1, dll_{MAP}).x_8.\\
&\quad \mathit{AuthenticationPackageLoaded}(u_1, sess_1, dll_{setuid}).x_9.\\
&\quad \mathit{PrivilegeServiceCalled}(u_1, sess_1, svr_{LSA}, sn_{Logon}, pv_{SeTcb}, s).x_{10}.\\
&\quad \mathit{TrustedLogonProcess}(u_1, sess_1, pn_{Wlogon}).x_{11}.\mathit{TrustedLogonProcess}(u_1, sess_1, pn_{MSGina})\\
&\quad .x_{12}.\mathit{PrivilegeServiceCalled}(u_1, sess_1, svr_{LSA}, sn_{Logon}, pv_{SeTcb}, s).x_{13}.\\
&\quad \mathit{TrustedLogonProcess}(u_1, sess_1, pn_{secD}).x_{14}.\\
&\quad \mathit{NotificationPackageLoaded}(u_1, sess_1, pkg_{scecli}).x_{15}.\\
&\quad \mathit{PrivilegeServiceCalled}(u_1, sess_1, svr_{LSA}, sn_{Logon}, pv_{SeTcb}, s).x_{16}.\\
&\quad \mathit{ProcessCreated}(u_1, sess_1, pn_{svcost}, pid_z, pn_{srvc}).x_{17}.\mathit{TrustedLogonProcess}(u_1, sess_1, pn_{dcom})\\
&\quad .x_{18}.\mathit{PrivilegeServiceCalled}(u_1, sess_1, svr_{LSA}, sn_{Logon}, pv_{SeTcb}, s).x_{19}.\\
&\quad \mathit{ProcessCreated}(u_1, sess_1, pn_{svcost}, pid_2, pn_{services}).x_{20}.\\
&\quad \mathit{ProcessTokenAssignment}(u_1, sess_1, pn_{srvc}, pn_{svcost}, pid_2, s).x_{21}.\\
&\quad \mathit{ObjectOpen}(u_2, sess_2, ot_{mo}, on_{srvc}, hid_x, pid_2, axs_{srvcConn}, s).x_{22} \looparrowright \varepsilon\rangle \mathtt{tt} \bigwedge\\
&\nu X\langle \mathit{SuccessfulLogon}(u_3, sess_3, lp_{ADV}, 5).x_{23}.\mathit{PrivilegeAssigned}(u_3, axs_{srvc}).x_{24}\\
&\quad \mathit{ProcessCreated}(u_1, sess_1, serviceName, pid_x, pn_{srvc}).x_{25}.\\
&\quad \mathit{ProcessTokenAssignment}(u_1, pn_{srvc}, sess_1, serviceName, pid_x, s).x_{26} \looparrowright \varepsilon\rangle \mathtt{tt} \bigwedge\\
&\nu X\langle \mathit{SuccessfulLogon}(u_2, sess_4, lp_{ADV}, 5).x_{27}.\mathit{PrivilegeAssigned}(u_2, axs_{srvc}).x_{28}.\\
&\quad \mathit{ProcessCreated}(u_1, sess_1, serviceName, pid_y, pn_{srvc}).x_{29}.\\
&\quad \mathit{ProcessTokenAssignment}(u_1, sess_1, pn_{srvcConn}, serviceName, pid_y, s).x_{30} \looparrowright \varepsilon\rangle \mathtt{tt}
\end{aligned}$$

Where:

u_1	= NT Authority\ System
u_2	= NT Authority\ Network Service
u_3	= NT Authority\ Local Service
dll_{LSASRV}	= c:\ windows\ system32\ LSASRV.dll:Negotiate
$dll_{kerberos}$	= c:\ windows\ system32\ kerberos.dll:kerberos
dll_{NTLM}	= c:\ windows\ system32\ msv1_0.dll:NTLM
$dll_{schannel}$	= c:\ windows\ system32\ schannel.dll:Microsoft Unified Security Protocol Provider
$dll_{WDigest}$	= c:\ windows\ system32\ wdigest:WDigest
dll_{MAP}	= c:\ windows\ system32\ msv1_0.dll:Microsoft_Authentication_Package_V1_0
dll_{setuid}	= c:\ windows\ system32\ setuid.dll:setuid
lp_{ADV}	= ADVAPI
axs_{srvc}	= {SeAuditPrivilege, SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege}
$axs_{srvcConn}$	= {READ_CONTROL, Connect to service controller, Lock service database for exclusive access}
svr_{LSA}	= NT Local Security Authority
sn_{Logon}	= LasRegisterLogonProcess()
pv_{seTcb}	= SeTcbPrivilege
pn_{wlogon}	= WinLogon
pn_{MSGina}	= WinLogon\ MSGina
pn_{secD}	= KSecDD
$pn_{svchost}$	= c:\ windows\ system32\ svchost.exe
pn_{srvc}	= c:\ windows\ system32\ services.exe
pn_{Dcom}	= DCOMSCM
pkg_{scecli}	= scecli
ot_{mo}	= SC_ManagerObject
$on_{services}$	= Services Active

B.2 Logon Model

The detailed explanation of the Logon process is found in Chapter 5, User Logon Steps. The sequence of functions required for this process are as follows:

$TrustedLogonProcess(NT\ Authority \setminus System, sees_1, Winlogon)$

$TrustedLogonProcess(NT\ Authority \setminus System, sess_1, Winlogon \setminus MSGina)$

$PrivilegeServiceCalled(NT\ Authority \setminus System, NT\ Local\ Security\ Authority/$
$\quad Authentication\ Service, LsaRegisterLogonProcess(), SeTcbPrivilege, s)$

$PrivilegeServiceCalled(NT\ Authority \setminus System, NT\ Local\ Security\ Authority/$
$\quad Authentication\ Service, LsaRegisterLogonProcess(), SeTcbPrivilege, s)$

$TrustedLogonProcess(NT\ Authority \setminus System, sess_1, KSecDD)$

$NotificationPackageLoaded(NT\ Authority \setminus System, scecli)$

$--\ this\ package\ will\ be\ notified\ of\ any\ password\ changes$

$PrivilegeServiceCalled(NT\ Authority \setminus System, NT\ Local\ Security\ Authority/$
$\quad Authentication\ Service, LsaRegisterLogonProcess(), SeTcbPrivilege, s)$

$<\ Here\ there\ can\ be\ quite\ a\ few\ events\ in\ between\ >$

$SuccessfulLogon(NT\ Authority \setminus Network\ Service, sess_2 Advapi, 5)$

$PrivilegeAssigned(NT\ Authority \setminus Network\ Service, \{SeAuditPrivilege,$
$\quad SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege\}, s)$

$LogonAttempt(NT\ Authority \setminus System, user,$
$\quad MICROSOFT\_AUTHENTICATION\_PACKAGE\_V1\_0, s)$

$SuccessfulLogon(user, sess_3, User32, 2)$

$PrivilegeAssigned(user, \{SeChangeNotifyPrivilege, SeBackupPrivilege, SeRestorePrivilege,$
$\quad SeDebugPrivilege\}, s)$

$ProcessCreated(NT\ Authority \setminus System, sess_1, C : \setminus windows \setminus system32 \setminus userinit.exe, pid_2,$
$\quad C : \setminus WINDOWS \setminus system32 \setminus userinit.exe)$

$ProcessTokenAssignment(NT\ Authority \setminus System, sess_1, c : \setminus windows \setminus system32\setminus$
$\quad winlogon.exe, c : \setminus windows \setminus system32 \setminus userinit.exe, pid_2, s)$

Using the formal framework, this is how the process will be expressed:

$$\begin{aligned}
&\langle\!\langle SEC\rangle\!\rangle\langle x_1.\mathit{TrustedLogonProcess}(u_1, sess_1, pn_{wlgn}).x_2.\\
&\quad \mathit{TrustedLogonProcess}(u_1, sess_1, pn_{MSGina}).x_3.\\
&\quad \mathit{PrivilegeServiceCalled}(u_1, sess_1, sess_1, svr_{LSA}, sn_{Logon}, pv_{SeTcb}, s).x_4.\\
&\quad \mathit{PrivilegeServiceCalled}(u_1, sess_1, sess_1, svr_{LSA}, sn_{Logon}, pv_{SeTcb}, s).x_5.\\
&\quad \mathit{TrustedLogonProcess}(u_1, sess_1, pn_{secD}).x_6.\mathit{NotificationPackageLoaded}(u_1, pkg_{scecli}).\\
&\quad x_7.\mathit{PrivilegeServiceCalled}(u_1, sess_1, sess_1, svr_{LSA}, sn_{Logon}, pv_{SeTcb}, s).x_8.\\
&\quad \mathit{SuccessfulLogon}(u_2, sess_2, lp_{adv}).x_9.\mathit{PrivilegeAssigned}(u_2, axs_{srvc}).x_{10}.\\
&\quad \mathit{LogonAttemp}(u1, u_x, lp_{MAP}, s).x_{11}.\mathit{SuccessfulLogon}(u_x, sess_x, lp_{u32}, 2).x_{12}.\\
&\quad \mathit{PrivilegeAssigned}(u_x, axs_{admin}, s).x_{13}.\mathit{ProcessCreated}(u_1, sess_1, pn_{userinit}, pid_x, pn_{winlogon}).\\
&\quad x_{14}.\mathit{ProcessTokenAssignment}(u_1, sess_1, pn_{winlogon}, pn_{userinit}, pid_x).x_{15} \nrightarrow \varepsilon\rangle \mathtt{tt}
\end{aligned}$$

Where:

u_1	= NT Authority \ System
u_2	= NT Authority \ Network Service
pn_{wlgn}	= Winlogon
pn_{MSGina}	= Winlogon\MSGina
pn_{secD}	= KSecDD
$p_{userinit}$	= c:\ windows \ system32\ userinit.exe
$pn_{winlogon}$	= c:\ windows \ system32\ winlogon.exe
svr_{LSA}	= NT Local Security Authority\Authentication Service
sn_{Logon}	= LsaRegisterLogonProcess()
pv_{SeTcb}	= SeTcbPrivilege
pkg_{scecli}	= scecli
lp_{adv}	= ADVAPI
lp_{MAP}	= Microsoft_Authentication_Package_V1_0
lp_{u32}	= User32
axs_{srvc}	= {SeAuditPrivilege, SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege}
axs_{admin}	= {SeChangeNotifyPrivilege, SeBackupPrivilege, SeRestorePrivilege, SeDebugPrivilege}

B.3 Initializing Services at Startup

When a service starts, as mentioned earlier, it has to start under an account to use its privileges. What should appear in the logs is a successful log on, followed by a privilege assignment for that log on. Then the service is executed and a process begin appears in the security log, followed by a service start which can be logged either in the application log, or the system log, depending on the service.

Windows being a multi-threaded operating system, some logged events might not appear in the exact sequence as they should. Through our experimentation we have noticed this and we demonstrate the four possible ways that this can be modeled.

$$
\begin{aligned}
&< SEC > SuccessfulLogon(NT\ Authority \setminus LocalService, sess_2Advapi, 5)\\
&< SEC > PrivilegeAssigned(NT\ Authority \setminus LocalService, \{SeAuditPrivilege,\\
&\qquad\qquad SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege\}, s)\\
&< SEC > ProcessCreated(NT\ Authority \setminus System, sess_1, serviceName, pid_x, c : \setminus windows\setminus\\
&\qquad\qquad system32 \setminus services.exe)\\
&< SEC > ProcessTokenAssignement(NT\ Authority \setminus System, sess_1, c : \setminus windows\setminus\\
&\qquad\qquad system32 \setminus services.exe, serviceName, pid_x, s)\\
&< APP > ServiceStart(serviceName)
\end{aligned}
$$

OR

$< SEC >$ $SuccessfulLogon(NT\ Authority \setminus Network\ Service, sess_2 Advapi, 5)$

$< SEC >$ $PrivilegeAssigned(NT\ Authority \setminus Network\ Service, \{SeAuditPrivilege,$
$SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege\}, s)$

$< SEC >$ $ProcessCreated(NT\ Authority \setminus System, sess_1, serviceName, pid_x, c : \setminus windows \setminus$
$system32 \setminus services.exe)$

$< SEC >$ $ProcessTokenAssignement(NT\ Authority \setminus System, sess_1, c : \setminus windows \setminus$
$system32 \setminus services.exe, serviceName, pid_x, s)$

$< APP >$ $ServiceStart(serviceName)$

OR

$< SEC >$ $SuccessfulLogon(NT\ Authority \setminus Local\ Service, sess_2, Advapi, 5)$

$< SEC >$ $PrivilegeAssigned(NT\ Authority \setminus Local\ Service, \{SeAuditPrivilege,$
$SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege\}, s)$

$< SEC >$ $ProcessCreated(NT\ Authority \setminus System, sess_1, serviceName, pid_x, c : \setminus windows \setminus$
$system32 \setminus services.exe)$

$< SEC >$ $ProcessTokenAssignement(NT\ Authority \setminus System, sess_1, c : \setminus windows \setminus$
$system32 \setminus services.exe, serviceName, pid_x, s)$

$< SYS >$ $ServiceStart(serviceName)$

OR

$< SEC >$ $SuccessfulLogon(NT\ Authority \setminus Network\ Service, sess_2, Advapi, 5)$

$< SEC >$ $PrivilegeAssigned(NT\ Authority \setminus Network\ Service, \{SeAuditPrivilege,$
$SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege\}, s)$

$< SEC >$ $ProcessCreated(NT\ Authority \setminus System, sess_1, serviceName, pid_x, c : \setminus windows \setminus$
$system32 \setminus services.exe)$

$< SEC >$ $ProcessTokenAssignement(NT\ Authority \setminus System, sess_1, c : \setminus windows \setminus$
$system32 \setminus services.exe, serviceName, pid_x, s)$

$< SYS >$ $ServiceStart(serviceName)$

Using the formal framework, this is how the process will be expressed:

$$\langle x_1.\langle\!\langle SEC\rangle\!\rangle SuccessfulLogon(u_2, sess_2, lp_{adv}, 2).x_2.\langle\!\langle SEC\rangle\!\rangle PrivilegeAssigned(u_2, axs_{srvc}, s).$$
$$x_3.\langle\!\langle SEC\rangle\!\rangle ProcessCreated(u_1, sess_1, pn_{sn}, pid_x, pn_{srvc}).x_4.x_5.$$
$$\langle\!\langle SEC\rangle\!\rangle ProcessTokenAssignment(u_1, sess_1, pn_{srvc}, pn_{sn}, pid_x, s).$$
$$\langle\!\langle APP\rangle\!\rangle ServiceStart(pn_{sn}).x_6 \nrightarrow \varepsilon\rangle \mathtt{tt}$$
$$\vee\langle x_1.\langle\!\langle SEC\rangle\!\rangle SuccessfulLogon(u_3, sess_3, lp_{adv}, 2).x_2.\langle\!\langle SEC\rangle\!\rangle PrivilegeAssigned(u_3, axs_{srvc}, s).$$
$$x_3.\langle\!\langle SEC\rangle\!\rangle ProcessCreated(u_1, sess_1, pn_{sn}, pid_x, pn_{srvc}).x_4.$$
$$\langle\!\langle SEC\rangle\!\rangle ProcessTokenAssignment(u_1, sess_1, pn_{srvc}, pn_{sn}, pid_x, s).x_5.$$
$$\langle\!\langle APP\rangle\!\rangle ServiceStart(pn_{sn}).x_6 \nrightarrow \varepsilon\rangle \mathtt{tt}$$
$$\vee\langle x_1.\langle\!\langle SEC\rangle\!\rangle SuccessfulLogon(u_2, sess_2, lp_{adv}, 2).x_2.\langle\!\langle SEC\rangle\!\rangle PrivilegeAssigned(u_2, axs_{srvc}, s).$$
$$x_3.\langle\!\langle SEC\rangle\!\rangle ProcessCreated(u_1, sess_1, pn_{sn}, pid_x, pn_{srvc}).x_4.$$
$$\langle\!\langle SEC\rangle\!\rangle ProcessTokenAssignment(u_1, sess_1, pn_{srvc}, pn_{sn}, pid_x, s).x_5.$$
$$\langle\!\langle SYS\rangle\!\rangle ServiceStart(pn_{sn}).x_6 \nrightarrow \varepsilon\rangle \mathtt{tt}$$
$$\vee\langle x_1.\langle\!\langle SEC\rangle\!\rangle SuccessfulLogon(u_3, sess_3, lp_{adv}, 2).x_2.\langle\!\langle SEC\rangle\!\rangle PrivilegeAssigned(u_3, axs_{srvc}, s).$$
$$x_3.\langle\!\langle SEC\rangle\!\rangle ProcessCreated(u_1, sess_1, pn_{sn}, pid_x, pn_{srvc}).x_4.$$
$$\langle\!\langle SEC\rangle\!\rangle ProcessTokenAssignment(u_1, sess_1, pn_{srvc}, pn_{sn}, pid_x, s).x_5.$$
$$\langle\!\langle SYS\rangle\!\rangle ServiceStart(pn_{sn}).x_6 \nrightarrow \varepsilon\rangle \mathtt{tt}$$

Where:

u_1	= NT Authority\ System
u_2	= NT Authority\ Local Service
u_3	= NT Authority\ Network Service
axs_{srvc}	= {SeAuditPrivilege, SeAssignPrimaryTokenPrivilege, SeChangeNotifyPrivilege}
pn_{srvc}	= c:\windows\system32\services.exe
pn_{sn}	= serviceName
lp_{adv}	= ADVAPI

B.4 Account Creation, Modification, and Deletion

To add a user, a handle must be given to access the domain to which the user will be added with access right to create a user, read the password parameter to change the password, and lookup account IDs to create a new ID. The account we create here is called *user* and once we create the account, we set a password immediately. So the log should indicate *user* being added to group *Elabeth*, which is the domain in this case. Following in the log there should be an account created, enabled, changed, and the password set. Once this is done, the handle which was opened for that purpose is closed. After the user has been created and the password set, the new password and user have to be registered in the *SAM_USERS* and *SAM_ALIAS* respectively, and the account is added to Users group, which is a default setting. Note that in some cases an object is opened and a handle is assigned to change an account, for example, but the change is logged right after the handle is closed. This is a normal behavior, some events are logged after the even, not while it's being done. In this case, the log indicates an account changed and a handle with access rights to perform the necessary changes. In such cases we use common sense.

$ObjectOpen(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, SAM\_DOMAIN$
$\quad Elabeth, hid_{762480}, pid_{872}, \{ReadPasswordParameter, CreateUser,$
$\quad LookupIDs\}, s)$

$GroupMemberChange(Elabeth \setminus Administrator, Elabeth \setminus user, Elabeth \setminus None, added)$

$AcountCreated(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, user, Elabeth)$

$AccountEnabled(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, user, Elabeth)$

$AccountChanged(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, user, Elabeth)$

$PasswordSet(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, user, Elabeth)$

$HandleClose(hid_{762480}, pid_{782})$

$AccountChanged(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, user, Elabeth)$

$ObjectOpen(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, SAM\_USER,$
$\quad Domains \setminus Account \setminus Users \setminus 0000458, hid_{831536}, pid_{872}$
$\quad \{READ\_CONTROL, WRITE\_DAC, WritePreferences, ReadAccount,$
$\quad WriteAccount, SetPassword(withoutknowledgeofoldpassword)\})$

$HandleClose(hid_{831536}, pid_{782})$

$AccountChanged(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, user, Elabeth)$

$GroupMemberChange(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, Elabeth \setminus user,$
$\quad Builtin \setminus Users, added)$

$ObjectOpen(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, SAM\_ALIAS,$
$\quad Domains \setminus Builtin \setminus Aliases \setminus 0000221, hid_{782480}, pid_{872}, \{AddMember,$
$\quad RemoveMember, ListMembers, ReadInformation\}, s)$

$HandleClose(hid_{78240}, pid_{872})$

After creating the user, we want to add him to the administrator group to have administrative privileges. In this case we should see in the log a object opened with privileges to add, remove, and list members as well as read information. Following this event we should see a user added to the group in concern. As mentioned earlier, the log might show that the handle has been closed before the group member added event, but we rely on the type of object added and the access it has since the sequence of events is not always accurate.

$$\begin{aligned}
&ObjectOpen(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, SAM\_ALIAS,\\
&\qquad Domains \setminus Builtin \setminus Aliases \setminus 0000221, hid_{820456}, pid_{872}, \{AddMember,\\
&\qquad RemoveMember, ListMembers, ReadInformation\}, s)\\
&HandleClose(hid_{820456}, pid_{872})\\
&GroupMemberChange(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, Elabeth \setminus user,\\
&\qquad Builtin \setminus Administrators, added)
\end{aligned}$$

So far we have seen how a user is created, setting the password, and giving administrative privileges or adding the user to a group of administrators. Now we will see what happens when a user is deleted. The deletion of a user is logically the reverse process of creating a user. When we created the user and added him to the administrators group, he was actually added to *Users* group, *Administrators* group, *Elabeth* domain, *SAM_USER*, and two places in *SAM_ALIAS*. The user has to be removed from all these locations before the account can be deleted. The process below depicts this account deletion:

$$\begin{aligned}
&OpenObject(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, SAM\_USER,\\
&\qquad Domains \setminus Account \setminus Users \setminus 0000458, hid_{985208}, pid_{872}, \{Delete\}, s)\\
&OpenObject(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, SAM\_ALIAS,\\
&\qquad Domains \setminus Builtin \setminus Aliases \setminus 0000221, hid_{1438088}, pid_{872}, \{RemoveMember\}, s)\\
&HandleClose(hid_{1438088}, pid_{872})\\
&GroupMemberChange(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”},\\
&\qquad \text{“}S-1-5-21-631774309-52733966-2344899341-112\text{”}, Builtin \setminus Users,\\
&\qquad removed)\\
&OpenObject(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”}, SAM\_ALIAS,\\
&\qquad Domains \setminus Builtin \setminus Aliases \setminus 0000220, hid_{1438088}, pid_{872}, \{RemoveMember\}, s)\\
&GroupMemberChange(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”},\\
&\qquad \text{“}S-1-5-21-631774309-52733966-2344899341-112\text{”}, Builtin\setminus\\
&\qquad Administrators, removed)\\
&HandleClose(hid_{1438088}, pid_{872})\\
&GroupMemberChange(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”},\\
&\qquad \text{“}S-1-5-21-631774309-52733966-2344899341-112\text{”}, Elabeth \setminus None,\\
&\qquad removed)\\
&AccountDeleted(Elabeth \setminus Administrator, \text{“}0x0, 0x200A7\text{”},\\
&\qquad \text{“}S-1-5-21-631774309-52733966-2344899341-112\text{”}, user, Elabeth)\\
&ObjectDeleted(hid_{985208}, pid_{872})\\
&HandleClose(hid_{985208}, pid_{872})
\end{aligned}$$

B.5 Policy Change

Some policy changes are very simple and straightforward especially since they are logged in a dedicated event indicating the change. For example, assume a user wants to get access to remotely login through terminal services. He has to open the Microsoft Management Console (mmc.exe) to get access to the local security settings. After which he will change the access. This is indicated through these two events:

$ProcessCreated(Elabeth \setminus Administrator,$ "$0x0, 0x200A7$"$, c: \setminus windows$
$system32 \setminus mmc.exe, pid_{5976}, c: \setminus windows \setminus system32 \setminus explorer.exe)$
$SysSecurityAccessGranted(Elabeth\setminus,$ "$0x0, 0x200A7$"$, Elabeth \setminus Administrator$
$SeRemoteInteractiveLogonRight)$

However, there are some policy changes that are not as obvious such as modifications to the security option: "Network Security: Do not allow anonymous enumeration of SAM accounts". To identify such a policy change we have to look for a process begin where the process name is *mmc.exe*. This process opens two objects, SAM_SERVER and SAM_DOMAIN. This is the same for all "Security Options" policy modifications:

$ProcessCreated(Elabeth \setminus Administrator,$ "$0x0, 0x200A7$"$, c: \setminus windows$
$system32 \setminus mmc.exe, pid_{5976}, c: \setminus windows \setminus system32 \setminus explorer.exe)$
$ObjectOpen(Elabeth \setminus Administrator,$ "$0x0, 200A7$"$, SAM\_SERVER, hid_{782480},$
$c: \setminus windows \setminus system32 \setminus lsass.exe, pid_{872}, \{DELETE,$
$READ\_CONTROL, WRITE\_DAC, WRITE\_OWNER,$
$ConnectToServer, ShutdownServer, InitializeServer, CreateDomain,$
$EnumerateDomains, LookupDomain, s)$
$ObjectOpen(Elabeth \setminus Administrator,$ "$0x0, 200A7$"$, SAM\_DOMAIN, hid_{762480},$
$c: \setminus windows \setminus system32 \setminus lsass.exe, pid_{872}, \{DELETE, READ\_CONTROL,$
$WRITE\_DAC, WRITE\_OWNER, ReadPasswordParameters,$
$WritePasswordParameters, ReadOtherParameters, CreateUser,$
$WriteOtherParameters, CreateLocalGroup, ListAccounts,$
$GetLocalGroupMembership, LookupID, AdministerServers\}, s)$
$HandleClose(hid_{782480}, pid_{872})$
$HandleClose(hid_{762480}, pid_{872})$

B.6 Firewall Startup

$< SEC > ProcessCreated(NT\ Authority \setminus System, sess_1, c : \setminus windows\setminus$
$system32 \setminus alg.exe, pid_1, c : \setminus windows \setminus system32 \setminus services.exe)$

$< SEC > ProcessTokenAssignment(NT\ Authority \setminus System, sess_1, c : \setminus windows\setminus$
$system32 \setminus services.exe, c : \setminus windows \setminus system32 \setminus alg.exe, pid_1, s)$

$< SYS > NeworkLocationAwarness(Service\ Control\ Manager, entered\ running\ state)$

$< SYS > RemoteAccessConnection(Service\ Control\ Manager, entered\ running\ state)$

$< SYS > ApplicationLayerGateway(Service\ Control\ Manager, sent\ a\ start\ control)$

$< SYS > ApplicationLayerGateway(Service\ Control\ Manager, entered\ running\ state)$

$< FWL > UDPReceive(192.168.130.1, 239.255.255.250, DROP, 1034, 1900, 153)$

$< SEC > FirewallPolicyLoaded(NT\ Authority \setminus System, NO, Standard, ON, Enabled,$
$Enabled, Enabled, Enabled, Disabled, Enabled, Enabled, ICMP_SET)$

$Where\ ICMP\_SET = \{Enabled, Disabled, Disabled, Disabled, Disabled,$
$Disabled, Disabled, Disabled, Disabled, Disabled, \}$

$< SEC > WFLApplicationException(NT\ Authority \setminus System, LocalPolicy, Standard,$
$appPath, Enabled, Allsubnets)$

$< SEC > WFLPortException(NT\ Authority \setminus System, LocalPolicy, Standard,$
$interface, portNumber, protocol, enabled, scope)$

$\langle x_1.\langle\langle SEC\rangle\rangle ProcessCreated(u_1, sess_1 pn_{alg}, pid_1, pn_{srvc}).x_2.$
$\langle\langle SEC\rangle\rangle ProcessTokenAssignment(u_1, sess_1, pn_{srvc}, pn_{alg}, pid_1, s).x_3.$
$\langle\langle SYS\rangle\rangle NetworkLocationAwareness(src_{SCM}, act_{run}).x_4.$
$\langle\langle SYS\rangle\rangle RemoteAccessConnection(src_{SCM}, act_{run}).x_5.$
$\langle\langle SYS\rangle\rangle ApplicationLayerGateway(src$

where:

u_1	= NT Authority*setminus*System
pn_{alg}	= c:*setminus*windows*setminus*system32*setminus*alg.exe
pn_{srvc}	= c:*setminus*windows*setminus*system32*setminus*services.exe
src_{SCM}	= Service Control Manager
act_{run}	= Entered running state
act_{start}	= Sent a start control

After the firewall starts up and the firewall policy is loaded, exceptions for applications and ports that need to connect to the network are loaded as well. First the set of permitted applications and ports are loaded, starting with the all applications followed by the ports.

It's important to note that is a policy is loaded at startup as a group policy, and any exception was added as a local policy, this is a suspicious event. It might mean a malicious user is trying to enable a restricted port or an application to connect for malicious purposes. Another issue of concern is the path of the loaded application. In Windows Vista, for example, if the application path was changed, the application will not be allowed to pass through the firewall and connect to the network. Hence the path of the application must me checked to see if it is a legitimate application or some trojan horse.

Appendix C

Registry Correlation: USB Mounting Related Keys

The USB key we use is a standard 256MB flash stick, so there is no advance drivers or libraries that need to run. When the USB device is plugged in, we will see in the security log that the process rundll.exe started (Event ID 592) and the process that created it was svchost.exe. As for the registry, the number of keys created and set is quite a numerous as shown below:

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USB
    \Vid_xxxxPid_xxxx \ GUID

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USB
    \Vid_xxxxPid_xxxx \ GUID \ LocationInformation

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USB
    \Vid_xxxxPid_xxxx \ GUID \ Capabilities

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USB
    \Vid_xxxxPid_xxxx \ GUID \ UINumber

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USB
    \Vid_xxxxPid_xxxx \ GUID \ Control

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USB
    \Vid_xxxxPid_xxxx \ GUID \ LogConf

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USB
\Vid_xxxxPid_xxxx \ GUID \ DeviceParameters

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USB
\Vid_xxxxPid_xxxx \ GUID \ HardwareID

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USB
\Vid_xxxxPid_xxxx \ GUID \ CompatibleIDs

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Services \ USBStor \ Enum

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USB
\Vid_xxxxPid_xxxx \ GUID \ Control \ ActiveService

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USB
\Vid_xxxxPid_xxxx \ GUID \ DeviceParameters

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USB
\Vid_xxxxPid_xxxx \ GUID \ DeviceParameters \ SymbolicName

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USBStor
\Disk&Ven_&Prod_USB_Drive&Rev_2 \ GUID&0

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USBStor
\Disk&Ven_&Prod_USB_Drive&Rev_2 \ GUID&0 \ Capabilities

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USBStor
\Disk&Ven_&Prod_USB_Drive&Rev_2 \ GUID&0 \ UINumber

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USBStor
\Disk&Ven_&Prod_USB_Drive&Rev_2 \ GUID&0 \ Control

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USBStor
\Disk&Ven_&Prod_USB_Drive&Rev_2 \ GUID&0 \ LogConf

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USBStor
\Disk&Ven_&Prod_USB_Drive&Rev_2 \ GUID&0 \ HardwareID

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USBStor
\Disk&Ven_&Prod_USB_Drive&Rev_2 \ GUID&0 \ CompatibleIDs

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USBStor
\Disk&Ven_&Prod_USB_Drive&Rev_2 \ GUID&0 \ LogConf \ BootConfig

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ USBStor
\Disk&Ven_&Prod_USB_Drive&Rev_2 \ GUID&0 \ DeviceParameters

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Control \ DeviceClasses
\CLSID

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ Storage
\RemovableMedia \ x&xxxxxxxx&x&RM

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ Storage
\RemovableMedia \ x&xxxxxxxx&x&RM \ Capabilities

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ Storage
\RemovableMedia \ x&xxxxxxxx&x&RM \ UINumber

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ Storage
\RemovableMedia \ x&xxxxxxxx&x&RM \ Control

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ Storage
\RemovableMedia \ x&xxxxxxxx&x&RM \ LogConf

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ Storage
\RemovableMedia \ x&xxxxxxxx&x&RM \ HardwareID

HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Enum \ Storage
\RemovableMedia \ x&xxxxxxxx&x&RM \ CompatibleIDs

HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion \ Explorer
\MountPoints2 \ CPC \ Volume \ {CLSID}

HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion \ Explorer
\MountPoints2 \ CPC \ Volume \ {CLSID} \ Data

HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion \ Explorer
\MountPoints2 \ CPC \ Volume \ {CLSID} \ Generation

HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion \ Explorer
\MountPoints2 \ {CLSID}

HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion \ Explorer
\MountPoints2 \ {CLSID} \ BaseClass

HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion \ Explorer
\MountPoints2 \ {CLSID} \ Shell

HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion \ Explorer
\MountPoints2 \ {CLSID} \ Shell \ Autoplay