Automated Windows Event Log Forensics

By

Rich Murphey

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2007 USA

Pittsburgh, PA (Aug 13th - 15th)

available at www.sciencedirect.com

journal homepage: www.elsevier.com/locate/diin

Digital Investigation

Automated Windows event log forensics

Rich Murphey

Applied Cognitive Solutions, Houston, TX, United States

Keywords:
Computer forensics
Windows XP
Event logs
Log analysis
Automation
Process models
Requirements
Tool development

ABSTRACT

This paper proposes methods to automate recovery and analysis of Windows NT5 (XP and 2003) event logs for computer forensics. Requirements are formulated and methods are evaluated with respect to motivation and process models. A new, freely available tool is presented that, based on these requirements, automates the repair of a common type of corruption often observed in data carved NT5 event logs. This tool automates repair of multiple event logs in a single step without user intervention. The tool was initially developed to meet immediate needs of computer forensic engagements.

Automating recovery, repair, and correlation of multiple logs make these methods more feasible for consideration in both a wider range of cases and earlier phases of cases, and hopefully, in turn, standard procedures. The tool was developed to fill a gap between capabilities of certain other freely available tools that may recover and correlate large volumes of log events, and consequently permit correlation with various other kinds of Windows artifacts. The methods are examined in the context of an example digital forensic service request intended to illustrate the kinds of civil cases that motivated this work.

1. Background

Consider for a moment responding to a forensic service request to identify computer user activity during a specific time period in a 60 GB Windows XP desktop hard drive, and then, upon examination, finding that the hard drive had been reformatted and Windows XP had been reinstalled just after the specified time period. Reinstallation is a particular concern in this case, because when the system is reinstalled, by default the first few gigabytes of data are overwritten. This may overwrite much of the previous master file table (MFT).

In this case, much of the meta data may be overwritten, and few, if any, intact files are likely to contain sufficient time stamps to reconstruct events during the period of interest. Any remaining artifacts that contain time stamps of interest are likely to reside in unallocated portions of the file system or, in special circumstances, within newly allocated files whose clusters have not been overwritten. In order to identify events during the period of interest, one might consider recovering various kinds of artifacts that contain time stamps by data carving unallocated portions of the file system and possibly swap or hibernation files. The choice of whether to recover a significant number of time stamped artifacts and event records of interest in a commercial forensic engagement may depend heavily on the feasibility of recovery methods.

In the author's experience, either project budget or time to completion is an influential factor. When budgets are less of a concern, time to completion is often critical. Either way, time to completion for various analysis options becomes a competing criterion for prioritizing alternatives within a proposed scope of work. The scope of work often consists of top ranked work items prioritized to maximize the business value of the combined results. Items having labor intensive procedures are often delayed or eliminated by being ranked outside of scope. Thus, automation of any item can directly influence how often an item is a feasible candidate for consideration for

E-mail address: rich@murphey.org

1742-2876/$ – see front matter © 2007 DFRWS. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2007.06.012

a scope of work. In a practical sense, automation may be a deciding factor regarding which methods may be performed at each phase in an engagement, and how often a given type of analysis is performed in general.

2. Motivation

The issues that motivate automation in digital forensics have been studied from many perspectives. Automation that contributes to verifiable accuracy and error rates may enhance overall effective quality (Jeyaraman and Atallah, 2006) or perceived suitability or contribute to scientific support for a determination of admissibility (de Souza Oliveira et al., 2002). Automation can also promote the addition of a tool into the forensic analyst's software toolkit and thus affect acceptance of the method by the community (Geiger, 2005). For these methods to be suitable to provide expert evidence, it is no less important that the methods (1) avoid subjective interpretation, (2) withstand peer review and publication, and (3) be accepted by the respective scientific community. A failure to withstand such scrutiny early on can cause a method's use to become restricted (Mocas, 2003). The impact of these issues on event recovery can be examined by considering event data recovery as a component process of event reconstruction.

How do these issues relate specifically to event recovery? The effectiveness of event data recovery can be assessed by evaluating recovery needs within the context of event reconstruction frameworks such as Zeitline, XIRAF, and Stallard's expert system for automated interpretation.

Zeitline is a tool for combining various kinds of events and sources, and grouping them into complex events (Buchholz and Falk, 2005). Buchholz suggests that Encase, the Sleuth Kit, and the Forensic Toolkit (FTK) ''focus mainly on evidence recovery'' and ''have limited abilities to further analyze the data that is recovered'' to the extent that correlating time stamps from various kinds of artifacts may require a great deal of human effort. Although it is possible to search log files, one cannot ''combine different sources and establish any kind of temporal or logical ordering among them''. When time to completion is critical, automation and integration of these tasks may determine whether they are performed.

Alink (2005) cites overwhelming volumes of data as an obstacle to anything other than automated feature extraction and cites diversity of forensic tool output formats as an obstacle to unified analysis. He proposes separating extraction and analysis processes and using extensible markup language (XML) to reduce tool integration cost. Stallard and Levitt (2003) proposes automation of common interpretations of correlated meta data using an expert system containing rules based on known invariant relationships. They cite the availability of qualified forensic experts as a limiting factor that motivated their work, and suggest that the system may assist in providing more reliable and less refutable deductions. To the extent that such a system can provide uniform coverage of the data in applying inference rules, the system may relieve examiner effort that could then be spent on higher level interpretations and reporting on the meaning of the interpretations in the context of the case. Stallard's prototype decision tree shows a scenario in some ways similar to the example case data described here, in that it relates a sequence of time stamps across disparate objects when the object's user permissions meet certain criteria. Automating away manual effort for such correlations might allow the examiner to provide greater business value in reporting the contextual impact and significance of the interpretations.

3. Problem statement

Given the specific motivations to automate recovery when time stamps in meta data of files and folders of interest are unavailable, one might consider recovering various kinds of artifacts that contain time stamps from unallocated portions of the file system. These artifacts might include: event logs, registry hives, Recycle Bin indexes, Internet History indexes, and shortcuts. Standard digital forensic toolkits such as Encase, FTK, ProDiscover, and Sleuthkit have in common capabilities to sort and filter items by time stamp. However such capabilities to sort and filter by time stamp focus almost exclusively on time stamps from file and folder meta data in the file system, such as from the MFT on NT5. Thus, collating, sorting, and correlating time stamps exclusively from other artifacts can be considered to be outside the scope of standard procedures and current certification programs that focus on these toolkits.

The way in which this example case study falls outside the scope of standard tools and procedures serves to illustrate several inherent challenges to: (1) recover and extract a large volume of artifacts for which manual recovery and preprocessing may be infeasible; (2) repair corrupted, fragmented or partially overwritten data; (3) collate data from hundreds or thousands of files and records, and then; (4) Correlate disparate types of artifacts by time, location, user, etc. In addition, lack of support for these tasks by the standard toolkits shifts some burden of accuracy and verifiability from the tool to the examiner. The examiner may wish to address two questions: how can one ensure that the extracted data are a true and accurate representation of the data in situ? How can one make this accuracy transparently verifiable?

4. Tool requirements

Tool requirements can be refined by examining them with respect to individual steps defined by various forensic process models. The more direct indicators of requirements are provided by phases that integrate directly with data recovery. In the XIRAF model, artifact processing is divided into separate extraction and analysis processes (Alink, 2005). In the Enhanced Integrated Digital Investigation Process (EIDIP) model, interpretation is further separated from analysis as a separate subsequent phase (Baryamureeba and Tushabe, 2004). Together, these two models define three phases beginning with artifact recovery: extraction, analysis, and interpretation. Within the extraction phase the artifact recovery process might further be sub-divided into four steps beginning with data carving: recover, repair, validate, and collate. As will be discussed in the following sections, the behavior of the logging service motivates repair, and the frequent recovery of duplicate records or logs motivates collating a unified nonduplicate final set for export to other tools. This motivation may apply in part to certain other artifacts as well. In the following sections, this series of four steps are examined with regard to methods, and then in turn to results.

Given steps within the process, how can each phase be made transparently verifiable, preferably by post-processing tools? When the tools extract data, they should identify where and how the data are an exact copy of data in situ. When they repair data they should identify where and how data have been repaired or altered. Where the data are repaired, two desirable properties for the tool are ''duplication integrity'' (Mocas, 2003) for the body of the log, and ''identifiable interference'', for the header, in which changes in the original data are made clearly identifiable. So, it would be desirable for a tool that repairs artifacts to report separately for each repair, the location and hashes of the altered and unaltered portions of the data, in order to clearly demonstrate how the exported data and the in situ source data differ. That is, the tool should report separately for each repair the portion of the data that was altered for repair, the hashes before and after repair, and the differences in the data. By providing these details, the tools may permit the examiner to more transparently demonstrate how the alteration affects error rates and interpretation of the results. In a more practical sense, such features in effect determine the bounds (standards, laws, resources) within which the tool can be used (Mocas, 2003).

5. Methods

To meet these requirements, consider a sequence of processing steps having an overall goal of correlating disparate types of artifacts by time, source, user, etc., after the artifacts have been obtained from unallocated portions of the file system. The sequence is: (1) recover (data carve) artifacts, (2) repair carved items as needed, (3) validate resulting items, and then, (4) collate each type of item. Having collated a number of artifacts of each type, consider a sequence of tasks having a goal of formulating a response to the service request, including: (1) correlate via common attributes, (2) interpret correlations in context, and (3) summarize relevant facts.

Within this context, this paper focuses specifically on the processing of NT5 event log records. Although a variety of tools are discussed, the purpose is not to survey the tools, but rather to address their use in a specific concrete context based on direct experience, and compare alternative tools as they may relate to concrete experience of others.

5.1. *Step 1 – recover*

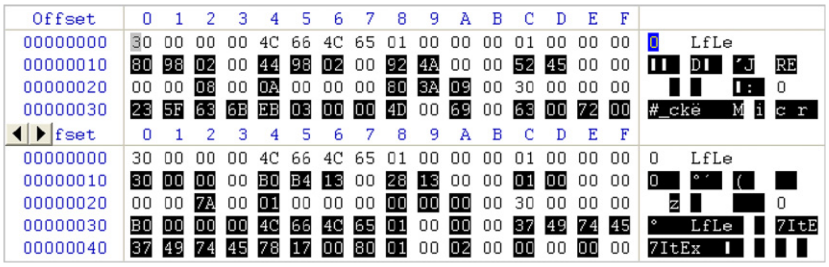
In order to data carve event logs from Windows NT5, several factors must be considered, including (1) the signature and other parameters required to carve a specific file format, (2) the utility to be used for data carving, and (3) what portion of the disk to carve. Carving utilities generally require, at a minimum, the signature of the file's header and a file length. The signature can be obtained by observing the differences between pairs of logs and noting the leading bytes that are identical between them, as indicated by the bytes that are not highlighted in the WinHex display shown in Fig. 1.

NT5 event logs have a fixed size that is configurable on a per-log basis. Although the default size of system event logs on NT5 is 512 KB, it is not unusual to see logs of 1 MB on workstations, and 16 MB on servers. Because the logging service memory maps the entire log file, there are practical limits on the size of event log files. According to developer documentation (MSDN, 2005) this limit is 300 MB for all the event logs combined on NT5; however, the performance impact of log file sizes of even 100 MB is generally a significant disincentive to using this log size. Based on this information the signature selected might be 0×300000004c664c650100000001000000, the minimum length might be 512 KB, and the maximum length 1 MB for workstations and 16 MB for servers.

Next, consider two open source data carving utilities, Foremost (Kendall and Kornblum, 2007) and Scalpel (Richard and Roussev, 2005), and for comparison, a commercial utility, File Extractor Pro (FEPro). Foremost and Scalpel have somewhat similar features and both are actively developed at the time of this writing. Scalpel was designed for lower memory usage and higher performance. The Scalpel 1.60 distribution includes a Win32 binary that can be downloaded and immediately run. Although Foremost is still under active development, Foremost 1.4 provides no Windows binary, and although it supports cross-compiling to Windows on OSX, it is not trivial to build.

Foremost and Scalpel configuration files have compatible syntax with respect to the parameters needed to carve NT5 event logs. The foremost.conf or scalpel.conf entry to carve the logs might consist of the scalpel.conf entry shown in Fig. 2 which specifies the output file extension 'evt', followed by 'y' indicating a case sensitive signature, the file length and the signature itself. For servers, the length might be 16777216 rather than 1048576.

Next, consider what portion of the disk to carve. This may depend in part on the nature of the source media, which can be a physical device, an evidence file mounted as a logical



```
 Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00000000  30 00 00 00 4C 66 4C 65 01 00 00 00 01 00 00 00   0   LfLe
00000010  80 98 02 00 44 98 02 00 92 4A 00 00 52 45 00 00   DI  ´J  RE
00000020  00 00 08 00 0A 00 00 00 80 3A 09 00 30 00 00 00     :  0
00000030  23 5F 63 6B EB 03 00 00 4D 00 69 00 63 00 72 00   #_ckë   M i c r
  fset     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00000000  30 00 00 00 4C 66 4C 65 01 00 00 00 01 00 00 00   0   LfLe
00000010  30 00 00 00 B0 B4 13 00 28 13 00 00 01 00 00 00   0   °´  (
00000020  00 00 7A 00 01 00 00 00 00 00 00 00 30 00 00 00    z        0
00000030  B0 00 00 00 4C 66 4C 65 01 00 00 00 37 49 74 45   °   LfLe    7ItE
00000040  37 49 74 45 78 17 00 80 01 00 02 00 00 00 00 00   7ItEx
```

Fig. 1 – Differing log headers.

```
evt y 1048576 \x30\x00\x00\x00\x4c\x66\x4c\x65
\x01\x00\x00\x00\x01\x00\x00\x00
```

Fig. 2 – scalpel.conf Entry.

device, or a raw image of a device. In the case of a physical or logical device, the documentation for Scalpel states that the supplied Windows executable does not support data carving a directly attached raw disk device using the Windows NT raw device name \\.\physicaldevice0. However, Scalpel can be built to do so using Cygwin (Noer, 1998), an open source UNIX like development environment for Windows that provides POSIX and UNIX compatible application programming interfaces that facilitates porting UNIX applications to Win32. When built with Cygwin, both Scalpel and Foremost can directly data carve a raw disk device using the POSIX device names that Cygwin provides. The POSIX device names required to specify the raw disk devices are shown in Table 1.

So, to data carve the 1st partition on disk 0, a common location for the system boot partition, the command line would be "scalpel /dev/sda1".

For comparison, the equivalent commercial tool for Windows, File Extractor Pro provides a graphical user interface and is bundled with a catalog of several hundred file signatures, and can carve just the unallocated portion of the file system directly from a raw disk device or partition. Although the open source data carving tools cannot do so directly, there are additional freely available tools, such as the Sleuthkit's 'dls', that can be used to export the unallocated space as files, and combined with Scalpel to carve only the unallocated space in several steps. The Sleuthkit Windows binary distribution is not built using Cygwin, and thus cannot, as it stands, directly access a partition of a raw disk device. In order to use Sleuthkit's 'dls' to export the unallocated portion of a file system, an examiner must supply an absolute sector offset of the beginning of the partition containing the file system. This usage of open source tools can be scripted with some effort, while the commercial tool cannot be scripted to function without user intervention.

Another practical consideration is that, for efficiency, data carving utilities are designed to carve multiple types of files in a single pass. So although the criteria for carving event logs may require only the unallocated space on the drive to be carved, in practice, the choice of what to carve is determined for a variety of artifacts. When carving Windows artifacts, this will often include shortcuts that contain a snapshot of file or folder time stamps that are small enough to be found within MFT entries. So when considered within the wider context of a process that correlates event log records with other artifacts, it may in general be desirable to carve the entire partition anyway, in which case the command above is sufficient.

Table 1 – POSIX disk device names

/dev/sda	Entire disk 0
/dev/sda1	Disk 0, 1st partition
/dev/sda2	Disk 0, 2nd partition
/dev/sdb	Entire disk 1
/dev/sda1	Disk 1, 1st partition

5.2. *Step 2 – repair*

Depending on the length of time the system has been in service, Section 5.1 may recover a significant number of logs. This may result from the way the log is emptied. When the event log is emptied, rather than truncating and overwriting the old log, the log is deleted and a new log created, often in another location on disk, leaving the old log intact in unallocated space. Low fragmentation rates typical of large hard drives may also contribute to successful recovery of complete logs. Finally, the length of time the system has been in service can contribute to the volume of recovered logs. The number of logs and volume of events were key issues that motivated automation of log repair. So, how can the nature of the corruption be determined and a way to repair hundreds of corrupted logs be devised?

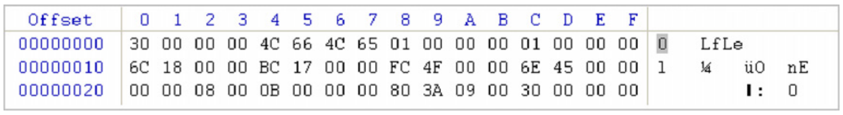
5.2.1. *How log corruption can occur*

The behavior of the Windows logging service may never be documented by the software vendor, because such specification might preclude further revisions in the encoding or system behavior. This may be the reason that there is no such documentation of logging service internals presently available for the Windows platform. With the absence of documentation, the service may instead be understood by leveraging system constraints and through observation. The motivation for discussing internals is that, regardless of how much the mechanism is understood, there may never be sufficient documentation available to defend the validity of repairing log files. Demonstration of a scientific basis as to the validity and appropriateness of these methods may, instead, depend on assigning fundamental design constraints to system behavior that, in turn, explain why these methods work.

Normally, when the event logging service shuts down, it saves the offsets of the oldest and newest events in a header at the beginning of the log file, and then it sets a 'clean' flag (borrowing file system terminology) in the header to indicate that these offsets are valid. Conversely, it sets the flag to 'dirty' while the service is running, appending event records to the log and not updating the offsets in the header. When the service appends new event records, rather than update these offsets in the header, it instead updates a duplicate set of offsets in a trailer immediately following the last appended record.

To better understand why the service behaves this way, the design criteria can be examined. One motivation for this behavior is better performance while writing event records. Updating the header every time a record is added to the log would require additional, discontinuous disk I/O that would reduce the rate of event throughput and might compete for I/O bandwidth with other applications. In order to enhance performance, the logging service avoids updating the header with each new log record. Instead, it maintains an up-to-date copy of valid offsets in the trailer, contiguous with the most recently appended record.

At various levels of hardware and software architecture, there are constraints such as cache line size, virtual memory



```
Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00000000 30 00 00 00 4C 66 4C 65 01 00 00 00 01 00 00 00  0   LfLe
00000010 6C 18 00 00 BC 17 00 00 FC 4F 00 00 6E 45 00 00  l   ¼   üO  nE
00000020 00 00 08 00 0B 00 00 00 80 3A 09 00 30 00 00 00          :  0
```

Fig. 3 – Corrupt log file header.

page size, file system cluster size, and disk sector size. I/O that fits within one size may reduce the need for further I/O to the next layer down. In this case, operating system and hardware context determines key design elements that are fundamental to understanding the nature of artifact encoding. If the event log offsets were maintained in any other location in the log, the performance cost for maintaining consistent offsets while appending records to the log would be greater.

When the log file is not closed properly, the 'dirty' flag will show that the header out of date, indicating that these offsets may be incorrect. Tools that read the event log, such as the Windows event viewer, must have a valid offset for the first record in the log in order to locate and read it. So the tools do not read the log unless the flag says the log is 'clean'. Because this form of corruption occurs frequently, the log often still contains another up-to-date copy of these offsets immediately following the most recent event record. This up-to-date copy can be used to repair the header, and that is what the tool presented in this paper does, as described below.

5.2.2. *A manual method of log repair*

In this section the work by Bunting to identify and manually repair corrupt event logs is discussed (Bunting, 2007; Anson and Bunting, 2007). He describes the signs of a common form of log corruption and a manual method to repair it using, for example, WinHex. Fig. 3 shows a binary display of the log header, which contains a flag byte that indicates whether the header is valid, together with a set of four 32 bit fields that contain the byte offset and the index number of both the newest and oldest records in the log.

These data may be replaced with redundant information located just after the trailing (oldest and most recent) log record. Bunting provides a signature, 0×111111112222222233333333344444444, that may be used to locate the trailer within the log (Fig. 4) that, in this form of corruption, contains valid replacement values for the four fields. To avoid confusion, note that this is a trailer that follows the last event record rather than a trailer at the end of the log file. He then describes a method of copying these values from the trailer into the header, and then setting the flag so that native applications can read the repaired log.

In addition to this information, it may be helpful to consider some software design criteria and performance issues involved in order to better understand the mechanisms and thus the circumstances with which the repair can be accurate.

5.2.3. *FixEvt – a tool for automated log repair*

FixEvt is a new tool that automates the repair of this form of corruption of Windows NT5 event logs (Murphey, 2007). FixEvt is command line application that performs this repair to each file specified on the command line in a single pass with no user intervention. To repair all the logs in the current directory, the command line would be ''fixevt *.evt''. So a script to carve logs (and presumably other artifacts as well) and then repair the logs would simply be:

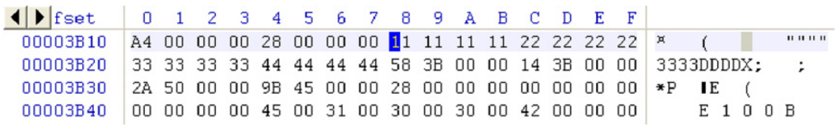
```
scalpel/dev/sda1
fixevt *.evt
```

FixEvt ignores the file if the flag indicates that the log is already clean. So invoking on logs that have already been repaired, or are not corrupt will have no effect.

When the flag indicates that the log is dirty, FixEvt scans the log for the trailer signature, 0×11111111222222223333333344444444. When the trailer is found, FixEvt copies the 16 bytes that immediately follow this signature into the header starting at byte 16 from the beginning of the log file, as shown in Fig. 5. Only the highlighted bytes in Fig. 5 have been altered from the original values shown in Fig. 3.

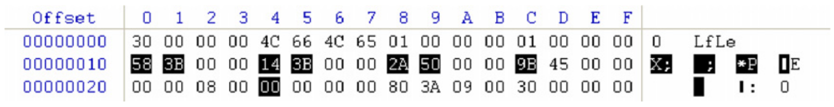
5.2.4. *Accuracy and error rates*

As discussed earlier, there is additional motivation to address accuracy and error rates in order to support perceived suitability and arguments for admissibility, and to support the examiner in reasoning about the effectiveness of applying the method in a specific context. First, the kind of data being repaired should be distinguished. For this form of corruption, the tool reads and writes only the meta data, not the contents of any individual event record. The automated repair described here is performed only upon the condition that the log is both marked 'dirty' and contains a trailer signature, and therefore contains replacement offsets to repair the header. The method alters a fixed location within the header, bytes 16 through 32. The rest of the log file is unaltered. So, the accuracy and error rates for the event records are unaltered in any way. The accuracy of the repaired meta data depends



```
Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00003B10 A4 00 00 00 28 00 00 00 11 11 11 11 22 22 22 22  ¤   (       """"
00003B20 33 33 33 33 44 44 44 44 58 3B 00 00 14 3B 00 00  3333DDDDX;   ;
00003B30 2A 50 00 00 9B 45 00 00 28 00 00 00 00 00 00 00  *P   E  (
00003B40 00 00 00 00 45 00 31 00 30 00 30 00 42 00 00 00      E 1 0 0 B
```

Fig. 4 – Log file trailer.



```
Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00000000 30 00 00 00 4C 66 4C 65 01 00 00 00 01 00 00 00  0   LfLe
00000010 58 3B 00 00 14 3B 00 00 2A 50 00 00 9B 45 00 00  X;  ;  *P  E
00000020 00 00 08 00 00 00 00 00 80 3A 09 00 30 00 00 00         :  0
```

Fig. 5 – Repaired log header.

entirely on the accuracy of the trailer. As discussed earlier, the trailer is written concurrently when appending records to the log, and this ensures that it is kept consistent at any given time with the records that have been written. Thus, this method of repair alters the header to be consistent with the records most recently appended to the log.

Note that by automating this process, the scope of data read and altered by this process is controlled and consistent. Automation ensures that the event records themselves are never altered, since such records are stored only at locations further in the file. However, additional validation could be performed on the header and event records to ensure that the tool can detect and then either report or reject invalid associated data which might affect the suitability of a repaired log for subsequent processing.

This paper addresses a single form of corruption, and does not address extraction of logs that are fragmented, partially overwritten, or corrupted in a more rare fashion. In practice any such corruption is identified in the validation step, and any corrupted logs are excluded from the final collation step.

5.3. *Step 3 – validate*

The number of recovered, repaired logs may complicate the process of characterizing what is present or absent in the logs, even within this context of a single host. So in general there may be a need to translate the records for additional processing in other tools. As a next step toward such processing it may be desirable to validate and collate the log records. For this we can consider an excellent free tool from Microsoft for analyzing Windows log files, called LogParser.

LogParser is free command line tool available for download from Microsoft (Giuseppini, 2005) that will run SQL queries on event log files. In addition to event logs, it can also read IIS, Exchange, and Snort logs, as well as the registry, file system, and active directory. It can output comma separated spreadsheets, XML, SQL, HTML, and others.

Each log may be validated by parsing all of the events in the log. The following SQL query parses the complete log as a result of counting the number of events in the log. If an error occurs, LogParser returns a nonzero status code to the shell. Conversely, if LogParser finds no errors while parsing the log, a zero status code is returned to the shell, and the next command then copies that validated log to a specified directory, 'goodlogs'.

```
for i in *.evt;do LogParser "select count(*) from $i" && cp $i
goodlogs; done
```

This puts a copy of all of the validated logs into the directory 'goodlogs.'

5.4. *Step 4 – collate*

Fourth, the logs may be collaged to obtain a single set of events that may be exported to other tools.

```
LogParser "SELECT * INTO allevents.tsv FROM goodlogs/*.evt
ORDER BY TimeGenerated"
```

There are often duplicate logs or logs that contain some overlapping set of duplicate event records. Duplicates may be an issue in various ways. Duplicate records may make it more difficult to count unique event occurrences. They may also conflict with common assumptions during analysis, or complicate the analysis. Finally, duplicates may conflict with requirements for unique records in post-processing tools such as databases.

If needed, duplicates may be filtered out by excluding the columns that specify the log file name and record number, and specifying the keyword 'DISTINCT'. Together with the first three steps, the complete script for the extraction phase might be as follows.

```
scalpel /dev/sda1
fixevt *.evt
for i in *.evt;do LogParser "select count(*) from $i" && cp $i
goodlogs; done
LogParser "SELECT DISTINCT TimeGenerated, EventID,
EventType, EventCategory, SourceName, SID, Message INTO
nonduplicates.tsv FROM goodlogs/*.evt ORDER BY
TimeGenerated"
```

Depending on what kind of tool integration is required, we might select output as XML, TSV (tab separated values), CSV (comma separated values), HTML template, syslog, or a direct ODBC connection to another database.

6. Results

The results of applying the above script can be examined by returning the case study and the examination of a 60 GB drive. As described earlier, the goal is to identify user activity during a specific time period that occurred prior to the hard drive being reformatted and Windows XP being reinstalled. In practice, many complete logs may be recovered from unallocated space. Depending on how long the system had been in service, it is not unusual to recover events for a year or more. So, the fictional but typical scenario might be as follows.

In order to perform extraction, run the above script on the write-blocked drive or the drive image. In the first step, scalpel obtains 100 1 MB data carved event log files. Of the 100 log files, event viewer will only open two of them, so only two out of 100 log files are demonstrably valid. In Section 5.2, FixEvt repairs 50 logs, for a total of 52 1 MB event logs that event viewer will open, and that appear to be valid. This volume of events may present a challenge, as discussed in Section 5.3. In general there may be a need to export the combined set of events to other tools. In Section 5.3, the script parses all the logs, identifies 46 that are valid, and moves these to the 'goodlogs' directory. Also as a result of Section 5.3, it is reported that half of the valid logs are empty or contain a single event, and the remaining 23 non-empty logs each contain an average of 3000 events. Combined, the 46 valid logs contain a total of 69,000 events spanning a period of 18 months prior to when the drive was reformatted. In Section 5.4, LogParser will in effect collate the 46 log files, and permits the filtering out of duplicate events to obtain a single, unified chronological view of the logs.

This is the result of the four-step extraction phase, and in turn, the input to the following analysis and interpretation phases.

In the analysis phase, the process models motivate integration with other frameworks and toolkits. However, integration is outside the scope of this paper, and thus only a few analysis steps are presented to illustrate the value of correlation that might be performed. A few simple queries will illustrate the utility of LogParser.

To obtain only the time and message for all nonduplicate events during a two-week period beginning 3/1/2006, the following query might be used.

```
LogParser "SELECT DISTINCT TimeGenerated, Message INTO 2006-03-01to03-15.tsv FROM goodlogs/*.evt WHERE TimeGenerated > '2006-03-01 00:00:00' AND TimeGenerated < '2006-03-15 00:00:00' ORDER BY TimeGenerated"
```

To get all messages during this period showing operation of the CD burning service, the following might be used.

```
LogParser "SELECT DISTINCT TimeGenerated, Message INTO 2006-03-01to03-15.tsv FROM goodlogs/*.evt WHERE TimeGenerated > '2006-03-01 00:00:00' AND TimeGenerated < '2006-03-15 00:00:00' AND Message LIKE '%CD-Burning%' ORDER BY TimeGenerated"
```

Using this query, the output shown in Table 2 might be obtained. This shows the CD burning service starting, running for several minutes, and shutting down.

The usefulness of this information can be illustrated by correlating it with other kinds of artifacts, based on time stamps. Table 3 shows a Windows shortcut (.lnk file) that contains a time stamp that coincides with the first event shown in Table 2. The contents of this shortcut file are essentially a snapshot of attributes of the file it refers to, a file that resides on D: which may be the CD drive. In fact, the shortcut shows that the device is specifically a CD-ROM. The volume label for the CD appears to be 'Nov 11 2006.' This is the default label that would have been created if the Microsoft CD burning wizard had used to burn the CD on that given day.

By correlating these two kinds of artifacts it appears that a file "OfInterest.doc" was burned to a CD labeled "Nov 11 2006" at 15:21 11/11/2006. If it were desirable to recover and identify this CD, the volume serial number could support additional verification as to whether the recovered CD was in fact the CD identified in the shortcut.

Table 2 – CD burning events

Time (UTC)	Message
11/11/2006 15:21	The CD burning service was successfully sent a start control.
11/11/2006 15:21	The CD burning service entered the running state.
11/11/2006 15:22	The CD burning service entered the running state.
11/11/2006 15:23	The CD burning service entered the running state.
11/11/2006 15:24	The CD burning service entered the running state.
11/11/2006 15:25	The CD burning service entered the running state.
11/11/2006 15:26	The CD burning service entered the running state.
11/11/2006 15:27	The CD burning service entered the running state.
11/11/2006 15:27	The CD burning service entered the stopped state.

7. Conclusions

This paper presents a method for completely automating the extraction phase of forensic processing of Windows event logs from a write-blocked drive or a drive image, in a single step without manual intervention. Within the extraction phase, the process model consists of four steps: recover, repair, validate, and collate. A script that automates the extraction process might consist of:

```
scalpel /dev/sda1
fixevt *.evt
for i in *.evt;do LogParser "select count(*) from $i" && cp $i goodlogs; done
LogParser "SELECT DISTINCT TimeGenerated, EventID, EventType, EventCategory, SourceName, SID, Message INTO nonduplicates.tsv FROM goodlogs/*.evt ORDER BY TimeGenerated"
```

In this script, the scalpel.conf may specify signatures for several artifacts in addition to event logs, then FixEvt repairs those logs that exhibit a common form of corruption, then those logs

Table 3 – Contents of shortcut (.lnk) file

Link target information	
Local path	D:\OfInterest.doc
Volume type	CD-ROM
Volume label	Nov 11, 2006
Volume serial number	E2C3-F184
File size	1643743
Creation time (UTC)	11/11/2006 3:21:14 PM
Last write time (UTC)	11/3/2006 10:12:34 AM
Last access time (UTC)	N/A

that show no parsing errors are moved to the directory 'goodlogs', and finally, LogParser culls duplicate records and provide a single chronological table suitable for subsequent analysis and interpretation in the file nonduplicates.tsv.

The utility of the results of automated extraction was illustrated with the description of a few simple analysis steps using SQL queries to efficiently search all of the recovered logs, using example searches constrained by time period and keywords. These steps illustrate the motivation for further automation of subsequent analysis, and even interpretation, and can be viewed as a requirement for tool integration with event reconstruction frameworks and forensic analysis toolkits. Even in the analysis and interpretation phases, at each step where SQL queries were applied to the results of the extraction phase, there remains a motivation for direct integration with other analysis and interpretation tools, and thus motivation to use widely supported encodings.

These methods focus on the use of the native platform (Windows) for extraction. Although vendor neutral tools and encodings are highly desirable (Carrier, 2002), the advantages of platform support for native encodings cannot be ignored. Vendors may consider such native artifact encodings to be an implementation detail for which there is a disincentive for documentation. There will probably always be tradeoffs between capabilities for depth of extracted details offered by native platform tools (Sardaukar) and integration offered by vendor neutral tools (Morgan). As a result of focusing on Windows artifacts, the methods presented here focus on using Windows as a host platform for doing the analysis. Geiger (2005) identifies two challenges to handling a multitude of native encodings: the numbers of applications that generate them and version skew with time. He suggests that tracking changes in encodings "would require considerable resources and sustained effort". This is in part the motivation for using native platform support that is addressed by breadth of features available in native Windows tools, including LogParser, for extracting and correlating such attributes. Altheide's (2004) survey of related tools for analysis on UNIX concludes that "One notable exception is the lack of a Linux-based Windows EVT viewer". There remains a significant number of native Windows tools relevant to both event log analysis and other artifacts that are either able to parse artifacts in more detail or parse a wider range of artifacts. This provides some motivation for porting other tools back to Windows using Cygwin, as has been done in the process of writing this paper.

The need for automation within the process models suggests that in order for these tools to be effective in assisting the examiner in performing event reconstruction, they should integrate with tools that focus on analysis and interpretation, such as Zeitline, XIRAF or Stallings's expert system. In particular, Zeitline's InputFilter interface is designed specifically for integration, so the work described here might easily be integrated with Zeitline. For this reason, the correlation of artifacts is not addressed in any further depth, but rather such discussion is left for future work involving integration with these frameworks. Integration that reduces manual effort required to pre- and post-process data at various steps of extraction, analysis, and interpretation in the XIRAF (Alink, 2005) and EIDIP (Baryamureeba and Tushabe, 2004) models may make the methods more suitable in general.

8. Future work

This work is being extended to NT6 (Vista) logs as well as other NT5 and NT6 artifacts. Shortly before submission of this paper, Bunting (2007) described an additional, though rare, way in which the event log can become corrupted that involve corrupt records in the middle of the log. The tool has been extended to detect and report this rarer form of corruption, and further work is being done to develop a method to perform automatically repair. As described in the accuracy and error rates section, there are motivations for implementing additional data validation of header and trailer information, if not records themselves. Finally, it is hoped that this tool or method may be integrated into event reconstruction frameworks or forensic analysis toolkits.

Acknowledgements

This paper extends Stephen Bunting's manual log repair method. Thanks to Captain Bunting for publishing this and other forensic methods and for offering additional challenges during the writing of this paper. The author wishes to thank Jerlyn Mardis at Applied Cognitive Solutions for editorial assistance and Matthew Geiger at CERT for valuable insight and review of drafts of this paper.

REFERENCES

Alink W. XIRAF – an XML information retrieval approach to digital forensics. Master's thesis, University of Twente, Enschede, The Netherlands; October 2005.

Altheide C. Forensic analysis of Windows hosts using UNIX-based tools. Digit Investig 2004;1:197–212.

Anson S, Bunting S. Mastering Windows network forensics and investigation. Wiley; 2007.

Baryamureeba V, Tushabe F. The enhanced digital investigation process model. In: Digital forensic research workshop; 2004.

Bunting S. Repairing corrupted Windows event log files, <http://128.175.24.251/forensics>; 2007.

Buchholz F, Falk C. Design and implementation of Zeitline: a forensic timeline editor. In: Digital forensic research workshop; 2005.

Carrier B. Open source digital forensics tools, the legal argument, <http://www.digital-evidence.org/papers/opensrc_legal.pdf>; 2002.

Carrier B. Sleuthkit and autopsy forensic browser, <http://www.sleuthkit.org>.

de Souza Oliveira F, Guimaraes CC, de Geus PL. Preforensic setup automation for Windows 2000, <http://www.las.ic.unicamp.br/paulo/papers/2002-CCN-IASTED-flavio.oliveira-PFSAF.pdf>; 2002.

Encase. Guidance software, <http://www.guidancesoftware.com>.

File Extractor Pro. StepaNet Communications, Inc, <http://www.datalifter.com>.

Forensic Toolkit (FTK). Accessdata, <http://www.accessdata.com>.

Geiger M. Evaluating commercial counter-forensic tools. In: Digital forensic research workshop, 2005, New Orleans, LA.
Giuseppini G, Burnett M, Faircloth J, Kleiman D. Microsoft Log Parser toolkit. Syngress; 2005.
Jeyaraman S, Atallah MJ. An empirical study of automatic event reconstruction systems. Digit Investig 2006;3S:S108–15.
Kendall K, Kornblum J. Foremost 1.4, <http://foremost.sourceforge.net>; 2007.
Morgan T. Grokevt, <http://www.sentinelchicken.org>.
Mocas S. Building theoretical underpinnings for digital forensics research. In: Digital forensic research workshop; 2003.
Murphey R, <http://www.murphey.org/fixevt>; 2007.
Noer G. Cygwin32: a free Win32 porting layer for UNIX applications. In: Proceedings of second USENIX Windows NT symposium. Seattle, Washington, August 3–4, 1998. <http://cygwin.com>.
Richard III GG, Roussev V. Scalpel: a frugal, high performance file carver. In: Proceedings of the 2005 digital forensics research workshop, New Orleans, LA. <http://www.digitalforensicssolutions.com/Scael>.
Stallard T, Levitt K. Automated analysis for digital forensic science: semantic integrity checking. In: 19th annual computer security applications conference, vol. 160; 2003.
Sardaukar. WindowsNT event log viewer, <http://www.codeproject.com/system/sysevent.asp>.
Technology Pathways. ProDiscover, <http://www.techpathways.com/>.
Threats and Countermeasures. Chapter 6: event log, <http://www.microsoft.com/technet/security/guidance/serversecurity/tcg/tcgch06n.mspx>; 2005.
WinHex 12.1, <http://www.winhex.com>.

Rich Murphey is a consulting forensic scientist with Applied Cognitive Solutions in Houston, TX. He received a Ph.D. in EE from Rice University before joining the faculty of Physiology and Biophysics at UTMB Galveston, then Halliburton as Sr. Scientist and then PentaSafe Security Technologies as Chief Scientist. He was a core team software developer for FreeBSD and XFree86.