Process Forensics: A Pilot Study on the Use of Checkpointing Technology in Computer Forensics

Mark Foster
Joseph N. Wilson
University of Florida

Abstract

The goal of this paper is to introduce a new area of computer forensics: process forensics. Process forensics involves extracting information from a process's address space for the purpose of finding digital evidence pertaining to a computer crime. The challenge of this sub-field is that the address space of a given process is usually lost long before the forensic investigator is analyzing the hard disk and file system of a computer. Therefore, the authors make the case that an accurate and reliable checkpointing tool could create a new source of evidence for the forensic investigator. The technology of checkpointing is nothing new when considering process migration, fault tolerance, or load balancing. However, with respect to computer forensics, the gains from checkpointing have yet to be explored.

Introduction

In recent years, computers and the Internet have become an integral part of our society. Computer are used in the workplace, home, school, and in some cases, public areas such as shopping malls and airports. The National Telecommunications and Information Administration (NTIA) released a report showing that Internet growth in the United States is estimated at 2 million new users each month [1]. The downside to this trend of pervasive computing is that the amount of computer-based crime is also on the rise. Statistics published by the CERT Coordination Center show the number of security related incidents have increased every year since 1998 [2]. From 2001 to 2003 the number of security related incidents more than doubled. As computer crime increases, so do the demands placed on computer security specialists and law enforcement.

To many computer security specialists, the idea of intrusion prevention is considered superior to that of intrusion detection. However, as long as intruders continue to be successful, the need for reliable intrusion detection systems is apparent. In addition, to prevent repetitive or similar intrusive attacks, reliable computer forensics are necessary to help determine why an attack occurred in the first place. Thus, computer forensics is an integral part of intrusion prevention.

The purpose of this paper is to introduce a new area of computer forensics. Currently, the discussion of this new area is largely theoretical. However, the authors' previous studies on checkpointing [3] and intrusion detection [4] have provided a unique perspective that gives them confidence that this new area can be far more than theoretical. Put simply, the unique perspective gained from studying both checkpointing and intrusion detection indicates that computer forensics is lacking in the sub-field termed *process forensics*. Process forensics involves extracting information from the process address space of a given program. This paper discusses how the information extracted from a process address space can be a source of evidence following a computer crime.

The collection of digital evidence in the form of process forensics can be divided into two areas. First, there must exist some tool that can extract information from a process address space. Secondly, one must know when to extract such information from a process address space. Checkpointing technology can be used for the extraction process. While the goal of checkpointing research is already aimed at storing key information about a process, the authors believe the proper checkpointing tool can also meet the needs of the evidence collector. Furthermore, intrusion detection systems can be utilized to help indicate when such information should be extracted from a process address space. Intrusion detection systems are already targeting the detection of malicious activity. It is malicious activity that is most likely to warrant the need for evidence surrounding a computer crime.

Background Overview

Checkpointing is the technique of storing a running process's state in such a way that a process can be restarted from the point at which the checkpoint was created. One creates a checkpoint by stopping the execution of a process, saving that process's address space and kernel state to a file, and then resuming the execution of that process. Saving this state to a file uses no additional system resources other than the stable storage necessary for storing a checkpoint file. Continued decreases in the price of disk storage make checkpoint storage cost effective. Well-known benefits of checkpointing include process migration, fault tolerance, and rollback recovery. Often, checkpoints are made at regular time intervals during the execution of a process. Use of storage space can be optimized when creating checkpoints at regular time intervals by saving only the difference between the current checkpoint and the most recent checkpoint. This is usually referred to as *incremental checkpointing*. In this paper, the concept of a *terminal checkpoint* is introduced. A terminal checkpoint is one that is created immediately prior to the termination of its associated process.

Stephenson defines *computer forensics* as the field of extracting hidden or deleted information from the disks of a computer [5]. Carrier [6] refers to computer forensics as the acquisition of hard disks and analysis of file systems. Simply put, computer forensics is the art of extracting digital evidence from a computer system usually associated with a crime. Not relevant to this discussion, computer forensics does at times include rescuing data from a damaged or corrupted computer system. Commonly, a computer forensic investigation takes place on the computer system that has either suffered an attack from another computer, or on the apprehended computer of a suspected criminal. In the case of the computer attack, the forensic investigator is usually attempting to find evidence that can answer questions such as, where did the attack originate, what vulnerability made the attack possible, and what files were compromised as a result of the attack. In the case of the suspected criminal's apprehended computer, the forensic investigator is usually looking for evidence of the suspected criminal's recent behavior, motives, or planning of future crimes.

In either case, the forensic investigator has a number of tactics for collecting such evidence. The investigation may involve anything from searching the file system for incriminating text files to analyzing log files for evidence of the attack. Typically, a computer forensic investigation involves using special forensic tools to analyze items such as slack space, unallocated space, or swap files. Slack space is the leftover space in a block or cluster allocated to a file but not used by the file. Unallocated space is space that is currently not used by any file. Both of these items may contain bytes from old files that were deleted, but have yet to be fully overwritten. This allows a digital forensic investigator to extract this data, using forensic tools. Swap files can be thought of as scratch paper for an application or the operating system. These files may have traces of data that allow the digital forensic investigator to piece together what actions have previously taken place on the given computer.

Another example of data often used in a forensic investigation that does not require special tools to extract is log files. During a forensic investigation, log files on the victimized or suspected criminal's computer are of the utmost importance. A survey of the literature on computer forensics reveals the direct correlation of logging and a successful forensic investigation [5,7,8]. Stephenson refers to the lack of logs as the single biggest barrier to a successful investigation of an intrusion.

Upon completion of a forensic investigation all of the extracted evidence is preserved and stored in a secure facility. A *chain-of-custody* is maintained to assure that no one tampers with the collected evidence. A chain-of-custody is simply a system of recording who is responsible for the evidence at any point in time from the moment it was collected till the moment it is used in a courtroom.

Slack space, unallocated space, swap files, log files, and most other items analyzed by the forensic investigator share an important similarity. Each of these items exists as nonvolatile data. Nonvolatile data has been saved to disk or resides on some other form of stable storage. Volatile data, on the other hand, resides in main memory such as a process's address space. Once a computer is unplugged from its power source, all volatile data is lost, but nonvolatile data remains intact. Due to the inherent nature of digital data, computer forensics is largely restricted to the analysis of nonvolatile data. One of the major keys to improving and enhancing computer forensics is to increase the amount of relevant nonvolatile data available to the forensic investigator. In this paper, the idea of using checkpointing technology to create additional nonvolatile data from one of the most common forms of volatile data, namely processes, is discussed. This would add checkpoint image files to the collection of items the forensic investigator can analyze for evidence.

Proposed Process Forensics

All work on a computer system is done in the form of a process. Processes can be divided into two categories: user-space processes and kernel-space processes. For the purpose of this discussion, only user-space processes are referred to. The reason for this is that a given kernel-space process is always acting on behalf of a particular user-space process or processes. Regardless of the unique methods different platforms used to handle processes, most all processes contain a great deal of information. Unfortunately, due to the nature of computer forensics, by the time a forensic investigation has begun, most of the relevant processes have already been terminated. Often, the involved computer system has been completely shutdown. The only data with which a digital forensic investigator can analyze processes are any files created while a given process was executing. Thus, the digital forensic investigator is left with a very limited amount of evidence concerning processes. Computer forensics can and should be expanded to include more process information. Checkpointing is one means to create more evidence to support process forensics.

The inspiration for applying checkpointing technology to the field of computer forensics stems from the authors' previous study on checkpointing [3]. From this study the authors gained valuable insights that led them to believe checkpointing has a role among the tools of the digital forensic investigator. Earlier checkpointing was referred to as the technique of storing a running process's state in such a way that a process can be restarted from the point at which the checkpoint was created. Put more simply, checkpointing creates a snapshot of a process's address space. This snapshot has the ability to restart the given process, thus it must contain all pertinent details about the process. The authors' study indicates that checkpointing does not have to be restricted to processes in a controlled environment. As the study shows, the development of run-time and transparent checkpointing tools is very feasible. Furthermore, checkpoints can be created very quickly, without modifying the process being checkpointed. This allows one to checkpoint malicious processes without affecting them and without notifying any potential attacker who is controlling such processes.

Possible Evidence in a Checkpoint

The discussion continues with a quick overview of the main sources of information that may be found in the checkpoint of a process. Recall that a process exists in main memory, where it has been assigned its own address space. Some of the more useful information found in a process's address space, and therefore included in a checkpoint, consists of items such as the process identification (PID) or the user who owns the given process, and pointers to parent, child, and sibling processes. While an attacker may have altered some of this information, it still provides a starting point for the forensic investigator. Information such as a PID is essential in distinguishing between multiple processes. Furthermore, knowing what user owned a process indicates who started the process or whose account may have been compromised. Ownership of a process, whether legitimate or not, also tells us the permissions level of the process. Clearly, a process run as root can do far more damage than a typical user process. In addition, knowing the relationship between different processes can assist in isolating the source of a process or what other processes resulted from the execution of a process. The parent and sibling relationship between processes is something not likely found in log files.

One of the more notable portions of a process's address space is the stack. The stack contains significant information pertaining to the execution sequence of a process. This sort of information is extremely useful to someone investigating a buffer overflow or *stack smashing* attack [4]. Given access to the stack, a digital forensics investigator can determine both where and how such an attack was made possible. Without knowing where and how an attack was made possible, it is very difficult to prevent similar future attacks without limiting one's usage of his own system. The process address space also contains the heap, bss, and data segments of a process. Analysis of the heap segment may reveal evidence pertaining to a *heap smashing* attack much like the stack in a buffer overflow or stack smashing attack. The stack, heap, bss, and data segments are all potential targets of malicious input attacks. In turn, each of these items would contain essential evidence of such an attack. As an integral part of the process address space, each of these items is included in a checkpoint image file. An additional example of this sort of malicious input attack would be a *format string attack*.

A process's address space also contains information about items referred to as process peripherals. Process peripherals include opened files, sockets, and pipes. Knowing what files a process accessed can be extremely valuable to the forensic investigator. This can indicate the intruder's objective, help isolate the damage done during the attack, or indicate attempts by the intruder to cover his or her own tracks. The digital forensic investigator and system administrator need to know if files such as password or log files have been modified or accessed. Tampering with a password file indicates the likelihood of future attacks via a compromised account. When log files have been tampered with, it usually indicates an attacker is attempting to cover his/her tracks. Socket connections provide additional evidence of communication links involved in a crime. Socket connections may indicate from where an attacker is launching an attack or where the attack is dumping stolen data. Pipes are another form of communication in which the digital forensic investigator would take an interest. Some checkpoints even include data that is still in the pipe buffer. Process peripherals could also include items such as a process's corresponding tty or terminal. With this information, the digital forensic investigator may learn whether or not the attack was launched locally.

The possibilities of evidence from process forensics are quite vast. However, it is essential to know when to collect process forensics in order to gain from it. The following section addresses this issue.

Opportunities for Checkpointing

A system administrator must make a number of tough decisions when dealing with an intruder. At times, a system administrator may become aware of an attack while it is in progress. This may be a result of the system administrator's own monitoring of the system or an alert issued by an Intrusion Detection System (IDS). The knee-jerk reaction to such a scenario is to kill all the processes related to the attack. While such an approach can be very effective in stopping the attack, it does little toward collecting evidence about the attack. Furthermore, such an approach is likely to tell the intruder that he or she was detected. Most of the time, one does not want the intruder to know he or she was detected until there is enough evidence to prove a crime took place and who committed it. For those reasons, when an intrusion is detected, whether by the system administrator or IDS, the immediate actions should include collection of evidence, or more specifically process forensic data, using incremental checkpoints that can be created without alerting the intruder. Once the intruder's session is ended, whether by the system administrator or by the intruder himself, the resulting checkpoints can provide crucial information about the attack.

A recent look at the ICAT vulnerability statistics shows a significant number of the CVE and CVE candidate vulnerabilities were due to buffer overflows. For the years 2001, 2002 and 2003 buffer overflows accounted for 21%, 22%, and 23% of the vulnerabilities respectively [9]. While much work has been done to detect buffer overflow attacks, to the knowledge of these authors, little has been done to enhance the ability to collect evidence resulting from buffer overflow attacks. Process forensics derived from checkpointing can help fill this void. Recall that a checkpoint contains the stack, heap, data, and bss segments of a process. In the case of a buffer overflow attack, creating checkpoints the moment the attack is detected, and even while the attack is in progress, will likely collect vital evidence. A forensic investigator can use this information to determine more closely how and when the intruder entered the system. A thorough analysis of the stack is likely to show what function contains the exploited vulnerability. Isolating the vulnerability is essential to preventing a similar attack in the future. In the case of a stack smashing attack, any code injected onto the stack may uniquely correspond to code that can later be found on the attacker's computer. The same is true for a heap smashing attack and a process's corresponding heap segment. While this alone does not prove anything, it does provide an additional corroborating stream of evidence. Any additional such evidence is desirable in the case of a legal setting. Stephenson [5] reminds us that it takes a "heap of evidence, to make one small proof."

ICAT's CVE and CVE candidate vulnerabilities classified as buffer overflow attacks are actually a subgroup of a much larger classification. This larger classification, known as *input validation errors*, accounted for 49%, 51%, and 52% of the CVE and CVE candidate vulnerabilities for the years 2001, 2002, and 2003 respectively. The idea of collecting evidence about a buffer overflow attack from a checkpoint is based on the concept that a buffer overflow attack stems from malicious input. Such input has no choice but to become part of a process's address space.

This approach to process forensics and evidence collection can be expanded far beyond buffer overflow attacks to include other input validation errors, such as a boundary condition error. While some boundary condition errors result from a system running out of memory, others may result from a variable exceeding an assumed boundary. Inspection of variables in a checkpoint file may reveal such an assumed boundary and expedite the process of closing a vulnerability once exposed. The very nature of attacks that exploit input validation errors automatically leave evidence in a process's address space. The potential for evidence and process forensics from checkpointing intruder related processes resulting from such vulnerabilities have yet to be explored.

Most intrusion detection systems can be categorized as misuse detection and anomaly detection. Misuse detection usually refers to those systems that utilize some form of signature or pattern matching to determine whether or not a process is part of an intrusion. Anomaly detection usually refers to those systems that attempt to define *normal* behavior so that processes can be categorized as normal or intrusive. Due to the inherent challenge in defining what is *normal* behavior, these systems often rely on some form of threshold to distinguish between normal and anomalous behavior. Markov Chain Model [10], Chi-square Statistical Profiling [11], and Text Categorization [12] are examples of such approaches to anomaly detection. The authors propose that such anomaly detection systems use checkpointing as an evidence collection technique for processes that are approaching or have passed the given threshold. Incremental checkpoints can be used to continually collect evidence of a process's behavior for any process that is considered anomalous or nearing anomalous. This would result in process forensics for those malicious processes that never quite reach the threshold and would usually go undetected. In addition, this would support forensic investigations of processes that do cross the threshold. Such forensics could expedite finding out why a process deviated from its normal behavior.

A common dilemma facing the computer crime investigator when entering a crime scene is whether or not to unplug the computer [7]. Any work by the criminal that resides in main memory is lost if the computer is unplugged. However, forensic analysis of a hard disk must always be performed on a copy rather than the original. In order to create a copy of the confiscated hard disk, the computer must eventually be powered off. Depending on the platform, Stephenson usually recommends directly unplugging the power source [5]. This avoids any booby-traps that may be triggered if the machine is not shutdown in a particular manner. Regardless of the manner by which a machine is shutdown, all of the volatile data such as a running process is lost. This illustrates another example of where additional evidence may be gained by using checkpointing. Prior to shutting down or unplugging a computer, relevant processes could be checkpointed. The resulting checkpoint files would allow the forensic investigator to analyze the running processes at a later time.

Additional Enhancements

If a computer crime ever reaches the courtroom, any evidence presented before the court must have been preserved through a chain-of-custody [5]. In other words, one must be able to verify with whom and where the evidence has been held since the moment it was collected. In the case of a checkpoint, the checkpoint resides in a file and can therefore be digitally fingerprinted immediately following its creation. In a courtroom, this digital fingerprint can be verified to show that the checkpoint file remains unaltered. A time and date stamp can also be included and verified with a digitally fingerprinted checkpoint file.

In addition, a checkpoint stored as a file can easily be transferred to a secure location, much like some logging systems. It is often recommended that logs be stored on a secure system separate from the system that generates the logs. These log files are also commonly stored in an encrypted format. These measures deter an intruder from altering log files to cover-up his or her unauthorized access to a system. Checkpoint files can be treated in the same manner. They can be stored on secure systems separate from where they were created. This prevents an intruder from modifying or destroying any evidence that is collected in the form of a checkpoint file.

Sommer has provided a good analysis of why intrusion detection systems fall short of providing quality evidence [8]. The authors propose that a checkpointing system should be developed separately from an ID system. During an attack, the ID system can trigger the checkpointing system to handle any intrusion related processes. This allows the ID research to focus on detection, rather than evidence collection. A checkpointing system, due to its inherent goal of recreating a process, is already aimed at collecting information about a process. This goal can be more easily combined with the goal of evidence collection. Furthermore, by allowing checkpointing systems to provide the evidence collection, the need for drastic modifications to existing ID systems is alleviated.

The format of a checkpoint file could be shared among multiple platforms. The format of a checkpoint file should be standardized similar to that of the ELF format used on Linux platforms. The standardization of checkpoint file formats would facilitate a common ground from which law enforcement, academia, and other researchers can work. This would facilitate the development of tools for working with and analyzing checkpoint files. In addition, standardizing any aspect of the forensic investigation aids in training future forensic investigators. Furthermore, standardization would assist in the acceptance of checkpoint evidence in legal proceedings. Likewise, standardization could further facilitate process migration among different platforms.

Carrier [6] has proposed a balanced solution to the open/closed source debate with regard to digital forensic tools. Carrier urges that digital forensic tools be categorized into tools for extraction and presentation. He proposes that extraction tools should remain open source, while presentation tools can have closed source. Such a balanced solution could easily be applied to checkpointing tools. The checkpoint/restart engine of a checkpointing system could remain open source. This allows researchers and the digital forensics community to validate the inner-workings of such checkpointing tools. Meanwhile, the presentation tools used for presenting, visualizing, and analyzing the data from a checkpoint file could remain closed source. It is likely that many individuals involved in the legal proceedings following a computer crime do not posses the necessary technical skills for understanding the data found in a checkpoint file. This provides ample opportunity for software developers to create presentation tools for checkpoint files. The goal of making complex checkpoint file data easily understandable would create ample competition for the private sector.

Summary

Researching both checkpointing and intrusion detection results in a unique perspective. This perspective is that computer forensics is lacking in the sub-field termed process forensics. Process forensics involves extracting information from the process address space of a given program for the purpose of evidence collection. Since computer forensics is restricted to nonvolatile data, to improve computer forensics new sources of nonvolatile data must be found. Since checkpointing creates nonvolatile data from processes, including checkpointing technology with intrusion detection systems can create a new source of nonvolatile data. In turn, increasing the amount of nonvolatile data increases the amount of forensic evidence available to the digital forensic investigator. Since this evidence comes from processes, it is appropriate to refer to it as process forensics. This paper has explored different sources and benefits of process forensics, one primary example being the evidence collected by an intrusion detection system enhanced with checkpointing technology.

Palmer [13] suggests that the future is likely to bring even tougher standards for digital evidence. Standardizing items such as the checkpoint file format used for process forensics can help meet these standards. Standardizing methods of evidence collection can help thwart some of the scrutiny placed on digital evidence in a courtroom setting. In addition, standardizing the checkpoint file format helps facilitate the training process of future digital forensic investigators. Lastly, it encourages the development of tools used to analyze checkpoint files for the purpose of process forensics.

In [5], Stephenson addresses the importance of reconstructing the crime scene. Anything less than the ability to recreate an entire process state may lead to holes in the evidence required to identify an attacker or prevent similar future attacks. Checkpointing provides the necessary level of detail to recreate an entire process. Although many attacks to date have not necessitated checkpointing, the needs of the past should not limit our preparedness for the future.

In closing, researchers and the digital forensics community must continue to find new sources of evidence following computer crimes. In many cases checkpointing technology can achieve such a goal.

About the Authors

Mark Foster is a Ph.D. Candidate in the Computer and Information Sciences and Engineering department at the University of Florida. He completed his Bachelors degree in Computer Science at Vanderbilt University. He is interested in many aspects of computing, but especially checkpointing, intrusion detection and computer forensics. Outside of computers his main hobbies are running and movies.

Joseph N. Wilson is an Assistant Professor at the University of Florida's Computer and Information Science and Engineering Department. He received his Ph.D. in Computer Science from the University of Virginia Department of Computer Science in 1985. His interests lie in Systems, Programming Languages, Computer Vision, and Image and Signal Processing.

References

[1] A Nation Online: How Americans are Expanding Their Use Of the Internet, retrieved May 10, 2004, from http://www.ntia.doc.gov/ntiahome/dn/ .

[2] CERT Coordination Center Statistics, retrieved May 10, 2004, from http://www.cert.org/stats/cert_stats.html .

[3] M. Foster, J.N. Wilson, *Pursuing the Three AP's to Checkpointing with UCLiK*, Proceedings for the 10th International Linux System Technology Conference, October, 2003.

[4] M. Foster, J.N. Wilson, S. Chen, “Using Greedy Hamiltonian Call Paths to Detect Stack Smashing Attacks,” to appear in the Proceedings of the 7th Information Security Conference, Palo Alto, CA, September, 2004.

[5] P. Stephenson, *Investigating Computer-Related Crime*, CRC Press, 1999.

[6] B. Carrier, *Open Source Digital Forensics Tools: The Legal Argument*, @Stake Research Report, October, 2002.

[7] A. Yasinsac, Y. Manzano, *Policies to Enhance Computer and Network Forensics*, Proceedings of the 2001 IEEE Workshop on Information Assurance and Security, West Point, NY, June, 2001.

[8] P. Sommer, *Intrusion Detection Systems as Evidence*, First International Workshop on the Recent Advances in Intrusion Detection, Belgium, September, 1998.

[9] ICAT Vulnerability Statistics, retrieved June 27, 2004, from http://icat.nist.gov/icat.cfm?function=statistics.

[10] N. Ye, *A Markov Chain Model of Temporal Behavior for Anomaly Detection*, Proceedings of the 2000 IEEE Workshop on Information Assurance and Security, West Point, NY, June, 2000.

[11] N. Ye, Q. Chen, S. M. Emran, K. Noh, *Chi-square Statistical Profiling for Anomaly Detection*, Proceedings of the 2000 IEEE Workshop on Information Assurance and Security, West Point, NY, June, 2000.

[12] Y. Liao, V. R. Vemuri, *Using Text Categorization Techniques for Intrusion Detection*, 11th USENIX Security Symposium, August, 2002.

[13] G.L. Palmer, *Forensic Analysis in the Digital World*, International Journal of Digital Evidence, Spring 2002, Volume 1, Issue 1.