THE JOURNAL OF DIGITAL FORENSICS, SECURITY AND LAW

Journal of Digital Forensics, Security and Law

Volume 14 | Number 3 Article 2

9-1-2019

Improved Decay Tolerant Inference of Previously Uninstalled Computer Applications

Oluwaseun Adegbehingbe
George Mason University, oadegbeh@gmu.edu

James Jones
George Mason University, jjonesu@gmu.edu

Recommended Citation

Adegbehingbe, Oluwaseun and Jones, James (2019) "Improved Decay Tolerant Inference of Previously Uninstalled Computer Applications," *Journal of Digital Forensics, Security and Law*: Vol. 14 : No. 3 , Article 2.
DOI: https://doi.org/10.15394/jdfsl.2019.1626
Available at: https://commons.erau.edu/jdfsl/vol14/iss3/2

IMPROVED DECAY TOLERANT INFERENCE OF PREVIOUSLY UNINSTALLED COMPUTER APPLICATIONS

Oluwaseun Adegbehingbe and James H. Jones Jr.
George Mason University
Fairfax, Virginia, USA
{oadegbeh, jjonesu}@gmu.edu

ABSTRACT

When an application is uninstalled from a computer system, the application's deleted file contents are overwritten over time, depending on factors such as operating system, available unallocated disk space, user activity, etc. As this content decays, the ability to infer the application's prior presence, based on the remaining digital artifacts, becomes more difficult. Prior research inferring previously installed applications by matching sectors from a hard disk of interest to a previously constructed catalog of labeled sector hashes showed promising results. This prior work used a white list approach to identify relevant artifacts, resulting in no irrelevant artifacts but incurring the loss of some potentially useful artifacts. In this current work, we collect a more complete set of relevant artifacts by adapting the sequential snapshot file differencing method to identify and eliminate from the catalog file-system changes which are not due to application installation and use.
The key contribution of our work is the building of a more complete catalog which ultimately results in more accurate prior application inference.

1. INTRODUCTION

Digital forensics investigations are often limited when digital artifacts have been intentionally or inadvertently deleted and partially overwritten. The current approach for dealing with this situation is to use forensic analysis tools to attempt to recover files through file signature matching and data structure analysis [1]. Another approach is to match individual file sectors to known content or to search for sub-sector strings (keywords) of interest. This approach ignores the inferential value of combining matched sectors and strings from multiple source files. A partial solution to this problem as proposed by Jones and his collaborators [2] compares matched sectors to a catalog of known multi-file artifact sets associated with specific software applications, then computes a weighted score over those matched sectors.

The approach used by Jones is limited by the fact that the sequential snapshot file differencing method used to build the

application-sector catalog is highly restrictive in its artifact selection. The files chosen for inclusion in the catalog are restricted to files with filenames or file paths matching a few selected keywords related to the application name, i.e., a white list. For example, only installed files whose file paths contain the keyword “firefox” are used in constructing a hash database for Firefox application. This approach results in the exclusion of sectors from certain files of interest whose file paths do not consist of the chosen keyword of interest. Jones and his collaborators, in their approach, produced a satisfactory result, but removed some number of file and sector artifacts which likely have inferential value. The need exists for a modified approach which produces a more complete set of artifacts for each application.

In this work, we propose a method for generating a more complete and accurate artifact set for each application included in the catalog. This method involves accurately isolating all artifacts that are generated directly or indirectly by an application during the application's lifecycle. If file system changes are monitored during an application's life cycle, it can be observed that not all the changes can be linked to the application. For instance, some of the activities are from the operating system (OS), and some might be due to user activities not related to the application of interest. In order to create a more accurate set of artifacts associated with an application, we need a means of distinguishing between application related and non-application related changes on the system's persistent storage. In this work, we extend the prior sequential snapshot file differencing method by isolating all application related file system changes in order to collect a larger but still accurate artifact set for the catalog construction. This is achieved by identifying and eliminating non-application-related file system changes and related artifacts collected while monitoring a file system for changes caused by an application's life cycle activities. In this work, we achieve this goal by adding a "do-nothing" branch to the artifact collection process. The "do-nothing" branch is simply a computer system running concurrently with the computer system being monitored for file system changes during application life cycle activities (Install, Open, Close, Uninstall, Close, Restart). The "do-nothing" system is left untouched while the "application-run" system gets an application installed, used, and uninstalled. In addition to the already reported file differencing performed between base and post activity images of the "application-run" branch, we also perform file differencing between the base and post activity images of the "do-nothing" branch. File system changes which appear in both difference sets are deemed to be not related to the application of interest. These file system changes can be removed from consideration for use in generating artifacts for the specific application catalog set. The key contribution of our work is improved catalog completeness and accuracy as compared to the previous file differencing approach for catalog construction, and the improvement in the accuracy of application inference based on the more complete catalog. We compare our new catalog sets to those previously reported, and we compare the inferential performance of our technique using known ground truth test images, including images from the M57 Patents Scenario data set [3]. We show that improving the completeness and accuracy of the catalog will produce improved application inference. This work is relevant for law enforcement, intelligence, digital forensics, and user-activity profiling.

2. RELATED WORK

Software applications such as word processors, spreadsheets, web browsers, media play-

ers, etc., are of interest to the digital forensics community as they are the instruments of user and system activity. Determining a user's application use is an active research area because different types of software allows a user to perform certain tasks that may be of investigative interest, e.g. a web browser can be used to access illegal content, hacking tools can be used to illegally access a computer system, etc. Forensic research regarding software applications often deconstructs or observes the application software to determine how it operates and what digital artifacts are created, modified, or deleted in different scenarios. Many applications have been studied in this manner and reported publicly, for example cloud storage software such as Dropbox [4], anti-forensic tools such as SecureClean [5], various browser artifacts, etc., while other analysis has been conducted on a one-time basis or not released publicly for other reasons.

Observation-based forensic analysis of application programs is typically implemented using one or both of two primary techniques: system monitoring and differential analysis. Both techniques identify system-level changes that an application makes during the application life cycle. Process Monitor [6] is a system monitoring tool which provides a unified view of the file system, Registry and process activities. It determines file system and Registry changes using common operating system Application Programming Interfaces (APIs).

Differential analysis is a process that compare two objects and reports the differences between them. The differential forensic analysis formula developed by Garfinkel [7] can be expressed as: $A———R > \$B$. "If A and B are disk images and the examiner is evaluating the installation footprint of a new application, then R might be a list of files and Registry entries that are created or changed" [7]. The objects that can be compared could be disk images, files, or network traffic capture files. The result of the differential forensic analysis is the report of changes between the two objects, e.g., additions, modifications or deletions to the file system. Garfinkel et al [Ibid.] authored differential forensic analysis tools, such as idifference.py, rdifference.py, idifference2.py. The idifference.py and idifference2.py tools take two disk images as input and report on the file system differences between them. The rdifference.py tool takes two offline Registry hive files as input and reports on the differences between them. When an application program gets uninstalled or deleted by the user, files associated with the application are deleted and the data storage areas associated with those files are deallocated by the file system. These application artifacts are now subject to decay or destruction because they can get partially or completely overwritten when the operating system reallocates their associated clusters for new data. When the deleted files remain intact, there are tools such as TestDisk [8] that can be used to recover the files. There are also file carving techniques [9], [10] that can be used to recover partially overwritten files. Other work [11] studied deleted file persistence in digital devices and media.

During the course of the forensic examination of a digital device, one goal is to explain what was found in the digital device, possibly in relation to a crime. Often a hypothesis is proposed to explain how a computer crime was committed, and analysis of the digital media is performed to obtain evidence that would support or refute such a hypothesis. This analysis may be intended to answer simple questions involving timelines, or user or application program activities involving a digital device. When one aspect of a digital forensic examination is to infer previously deleted application program software on a digital device, current approaches and research can be grouped into one or more of the following categories, discussed in the sections

that follow: string and keyword searching, log analysis, file carving, hash-based carving, and matching.

A simple approach that can be used to infer previously uninstalled application on a persistent storage media is by parsing the media for certain strings or keywords related to the application program activities in a manner similar to that used for memory [12], registry [13] and network logs [14]. Tools such as EnCase [15] are used to search digital media for specific keywords without the need to parse the file system. The problem with this approach is that other information such as time stamps, file meta-data, etc., have to be used to interpret the extracted keywords to determine whether application program activities were responsible for the existence of those keywords on the media.

Another approach to infer previously uninstalled applications on digital media is analyzing log files [16] to find evidence of previously recorded application program activities. Application program activities can be logged by the application itself (for debugging or troubleshooting purpose), another interacting application, or the operating system. These log files may persist even after the application program has been uninstalled, and may serve as evidence of application program activities on the digital device. This approach works if the desired log exists and has not been tampered with, but is less useful if such log files don't exist or have been corrupted or tampered with [17].

All the approaches described above rely on artifacts with human-readable content like log entries, keywords, etc. Research work by GG Richard III [18] showed that unallocated cluster data and non-human-readable file content may also indicate the prior existence of a file, e.g., an application program, on the storage media. File system references for the "deleted" files are not necessarily needed to retrieve the application file fragments from a storage media. The extraction of unallocated files and file fragments is known as file carving. File carving is the process of recovering file contents from digital media without the help of a file system. In order to successfully extract complete files from a digital media, knowledge of the file format (e.g. file headers and footers) is necessary. Earlier file carving approaches only work with unfragmented file clusters which are in order. Advanced recent file carving approaches work even if the application files are made of multiple fragments file clusters [19] by using cluster classification techniques to identify clusters belonging to the same file.

When complete file recovery is not possible due to decay of a deleted file's contents, a sub-file forensics approach [20] is needed. Research has shown that one can prove file existence and hence application program activities on a digital device through a process called hash-based carving. Hash-based carving is a technique for detecting the presence of specific files on digital media by evaluating the hashes of individual data blocks, rather than the hashes of entire files. Hash-based carving has been successful [21] in identifying files that are fragmented, incomplete, or partially modified. In order for this approach to be used to successfully identify a specific file, a catalog or database of block hashes derived from the file of interest has to be pre-built. The catalog is then used to scan a test media in search of matching hashes. The higher the number of matching blocks, the higher the likelihood that the full file was previously present in the test media.

The sector matching and aggregation approach, proposed by Jones, is a means of inferring the likelihood that an application was previously installed on a examined system by matching sectors on the examined system with known stored sectors associated with the multiple files from the application of interest. From the matched sectors, poten-

tially probative sector blocks are selected and weighted using an inverse weighting scheme (ala Inverse Document Frequency (IDF)) to compute the inferential value of the matched sectors. The results from this technique were promising in that the technique was able to indicate past application activity even after the application has been uninstalled and the host system rebooted and used. Disk images from the M57 data set [3] were used to evaluate this approach. While the approach was able to identify previously uninstalled applications on the test images, it was determined that the approach could be improved. Specifically, the process of selecting the sectors that get stored in the catalog was quite coarse, relying on the sector belonging to files with keywords of interest in its filename or file path. We are improving this approach by implementing a better selection process to build the catalog.

3. APPROACH AND METHODOLOGY

Our approach, summarized in Figure 1, seeks to improve on the technique used by Jones. Where Jones reduced noisy sectors using a keyword white list approach, we do so by eliminating sectors that are obtained from systems running without any activities associated with the application of interest. This is our so-called “do nothing” branch.

Our initial catalog was built by file differencing multiple virtual machine disk images taken while installing, using, and uninstalling applications in a controlled environment. The virtual disk images obtained were processed to identify new, modified, or deleted files

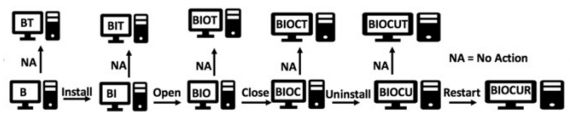

Figure 1. Approach Overview

between snapshots during the application install(I), open(O), close(C), uninstall(U) and system reboot (R) stages as indicated in Figure 1. The purpose of this process is to extract all the forensics artifacts an application of interest will create on a computer system throughout the application’s life cycle. Once extracted, the disk images are then analyzed with custom tools based on elements of the DFXML toolset [7] to generate a Digital Forensic XML (DFXML) file showing new, modified, or deleted files.

In our work, we also collect artifacts generated when the application of interest is not installed to identify artifacts due to non-application related activities (e.g. user, other application, or operating system related activities). Rather than restrict sectors of interest to files with filenames or file paths associated with application keywords, we include in the catalog all file system artifacts except those appearing in the do nothing branches. With the understanding that operating system activities occur simultaneously during the application software life cycle (i.e. install, open, close, uninstall and restart), we identify those artifacts in order to exclude them from the final catalog hash database. We achieve this by having a "do-nothing" branch that runs concurrently with the "application-run" branch. The "application-run" branch is a sequence of virtual machine snapshots taken after the occurrence of each part of the application software life cycle (install(I), open(O), close(C), uninstall(U) and system reboot (R)). The "do-nothing" branch is a sequence of virtual machine snapshots taken from virtual machines cloned from the "application-run" branch and run undisturbed and concurrently with the "application-run" branch. The purpose of the "do-nothing" branch is to collect file system activities that occur without the involvement or influence of the application of interest. These collected operating system related file system activities, if found

among file system activities observed in the "application-run" branch, will be excluded, resulting is a smaller and more accurate artifact set associated with the application of interest. The procedure described above and further explained in the subsequent subsections, is depicted in greater details in Figure 2.

We have two copies of the same base virtual machine instance (the two side-by-side boxes labeled “B”), one designated for the application install, use, and removal, the other designated for the "do-nothing" branch where the virtual machine (VM) is allowed to run concurrently with the “application-run” branch. The “do nothing” VM does not have the application of interest installed and has no user initiated activity. Immediately after the completion of each part of the application software life cycle, snapshots of the VMs at the "application-run" and "do-nothing" branches are taken. The VM at the "application-run" branch is then copied for use in the "do-nothing" branch in the next part of the experiment. This experiment is repeated, in sequential order, going through the stages of the application’s life cycle (Base-Install- Open-Close-Uninstall-Restart). After each action (e.g., Install), the “application-run” and “do-nothing” VMs are suspended and the respective disk images are archived for further processing in the next stage of the experiment. The “application-run” VM is cloned and the two VMs are set up for the next action (e.g. Open) with one designated for the “application-run” and the other designated for the “do- nothing” run. The next action (e.g. Open) is performed in the “application-run” VM while the “do-nothing” VM runs undisturbed. These steps are repeated as depicted in Figure 2 until all the actions in the life cycle for the application software are completed. At each step in the experiment, disk images are archived from the suspended VMs. The following subsections describe the experiment in greater detail.

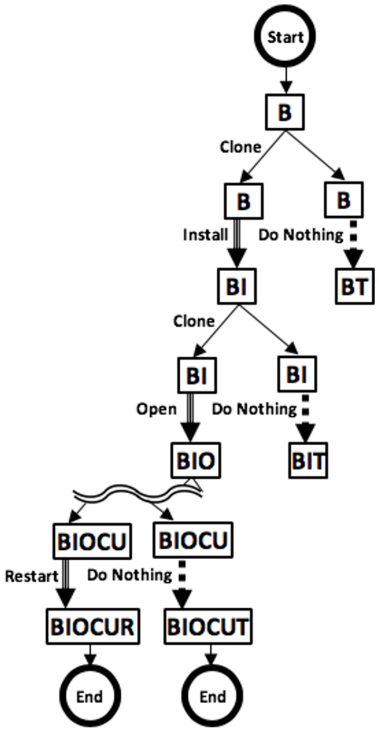

Figure 2. Overview of the VM buildup procedure for "application-run" and "do- nothing" branches with respect to catalog creation.

In this research effort, a catalog was created for 16 Windows applications in a controlled environment using virtual machine snapshots. These applications are the same applications selected in the NIST Diskprinting effort [22]. The 16 applications’ lifecycle were run in three Windows operating systems (Windows XP, Windows 7 32-bit, Windows 7 64-bit) to generate 29 application-OS combinations known as diskprints. Application-related files created during application Install, Open, Close, Uninstall and system Reboot are identified and associated sector and file information collected for ingestion into the catalog after some post-processing actions.

3.1 Build the Virtual Machine (VM) instances

1. Build the first VM by installing the operating system on the VM instance, adding the appropriate service pack installations so that the application software can successfully run within the VM instances.

2. BASE: The built base VM currently named "B" is cloned, with one copy designated for the “application-run” and the other designated for the “do-nothing” run.

3. INSTALL: The two VMs are to be run simultaneously. In the "application-run" VM, the application software of interest is installed while the "do-nothing" VM is allowed to run undisturbed for a period of time without user interaction of any kind. At the end of the application software installation, both VMs are suspended, the "do- nothing" VM is renamed as "BT" (Base-Time) and the "application-run" VM is renamed as "BI" (Base-Install) to reflect the current state of the VMs pertaining to the application life cycle stages. The virtual disk images in "BT" and “BI” are converted into raw disk images named "BT.img" and "BI.img" respectively.

4. OPEN: The VM currently named "BI" is cloned. In the "application-run" VM, the application software of interest is launched or run and used while the "do-nothing" VM runs undisturbed. At the end of the application software use, both VMs are suspended, the "do-nothing" VM is renamed as "BIT" (Base-Install-Time) and the "application-run" VM is renamed as "BIO" (Base- Install-Open). The virtual disk images in the two VMs are converted into raw disk images.

5. CLOSE: The VM currently named "BIO" is cloned. In the "application-run" VM, the application software of interest is closed or terminated while the "do-nothing" VM runs undisturbed. At the end of the application software exit, both VMs are suspended and renamed as "BIOC" (Base-Install-Open-Close) and "BIOT" (Base- Install-Open-Time) respectively. The virtual disk images in the two VMs are converted into raw disk images.

6. UNINSTALL: The VM currently named "BIOC" is cloned. In the "application-run" VM, the application software of interest is uninstalled or deleted while the "do-nothing" VM runs undisturbed. At the end of the application software uninstall, both VMs are suspended and renamed as "BIOCU" (Base-Install-Open-Close-Uninstall) and "BIOCT" (Base-Install-Open-Close-Time) respectively. The virtual disk images in the two VMs are converted into raw disk images.

7. RESTART: The VM currently named "BIOCU" is cloned. The "application-run" VM is restarted while the "do-nothing" VM runs undisturbed. At the end of the VM restart, both VMs are suspended, the "do-nothing" VM is renamed as "BIOCUR" (Base-Install-Open-Close- Uninstall-Restart) and "BIOCUT" (Base-Install-Open-Close-Uninstall-Time) respectively. The virtual disk images in the two VMs are converted into raw disk images.

3.2 Convert the VMWare VMDK file to a raw image file

1. The virtual disk images in the generated VMs are converted into raw disk images. Before the conversion, if the virtual

disk images are not flat (e.g., the disk is made up of differntial snapshots, like VDisk.vmdk, VDisk-s001.vmdk, VDisk-s002.vmdk, VDisk-s003.vmdk, etc.), the separate VMDK files would have to be combined into a single VMDK file using the VMWare's vmware-vdiskmanager tool as follows:$ vmware-vdiskmanager -r VDisk.vmdk -t 0 BIOCUR.vmdk

2. The resulting single VMDK file is then converted into a raw disk image file using the qemu-img tool as follows: $ qemu-img convert BIOCUR.vmdk -0 raw BIOCUR.img

3.3 Compare adjacent images using the idifference tool

The next step is to compare raw disk images and determine file system changes that have occurred between adjacent images or adjacent states in the "application-run" or "do- nothing" branch. For instance, comparing "B.img" to "BT.img" would allow us to determine the file system changes that occurred when the base VM is allowed to run for a period of time. The file system changes are stored in "B-BT.dfxml" file. Similarly:

1. "B.img" is compared with "BI.img" to identify file system changes due to application software installation. The file system changes are stored in "B-BI.dfxml" file.

2. "BI.img" is compared with "BIT.img" to identify file system changes due to the operating system in "BI" state running undisturbed. The file system changes are stored in "BI-BIT.dfxml" file.

3. "BI.img" is compared with "BIO.img" to identify file system changes due to the specific application software being launched or executed and use.. The file system changes are stored in "BI-BIO.dfxml" file.

4. "BIO.img" is compared with "BIOT.img" to identify file system changes due to the operating system in "BIO" state running undisturbed. The file system changes are stored in "BIO-BIOT.dfxml" file.

5. "BIO.img" is compared with "BIOC.img" to identify file system changes due to the launched or running application software getting terminated. The file system changes are stored in "BIO-BIOC.dfxml" file.

6. "BIOC.img" is compared with "BIOCT.img" to identify file system changes due to the operating system in "BIOC" state running undisturbed. The file system changes are stored in "BIOC-BIOCT.dfxml" file.

7. "BIOC.img" is compared with "BIOCU.img" to identify file system changes due to the application software getting uninstalled. The file system changes are stored in "BIOC-BIOCU.dfxml" file.

8. "BIOCU.img" is compared with "BIOCUT.img" to identify file system changes due to the operating system in "BIOCU" state running undisturbed. The file system changes are stored in "BIOCU-BIOCUT.dfxml" file.

9. "BIOCU.img" is compared with "BIOCUR.img" to identify file system changes due to the operating system in "BIOCU" state getting restarted. The file system changes are stored in "BIOCU-BIOCUR.dfxml" file. These comparisons are depicted in Figure 3.

The tool used to compare two raw disk image files is the idifference tool [7]. The idifference tool is a Python program that compares two raw image files and report the differences on the file objects that they contain. It reports on file system changes such as files deleted, files created, files moved or renamed, or files modified. The output is a DFXML file. The general command used for comparing the two raw image files (B.img and BT.img) as follows: $ python idifference2.py -x B-BT.dfxml B.img BT.img

3.4 Shrink the DFXML files and generate a JSON file containing digital artifacts

One of the motivations of this research experiment is to eliminate all file system changes attributed to operating system or other application activities from the digital artifacts that would be ingested into the catalog database. For instance, "B-BI.dfxml" supposedly represents file system activities due to application software installation. However, while the application software program was being installed, there are other file system changes occurring simultaneously that are actually due to operating system activities and not due to the application installation. These operating system-related activities must be removed from our current data sets.

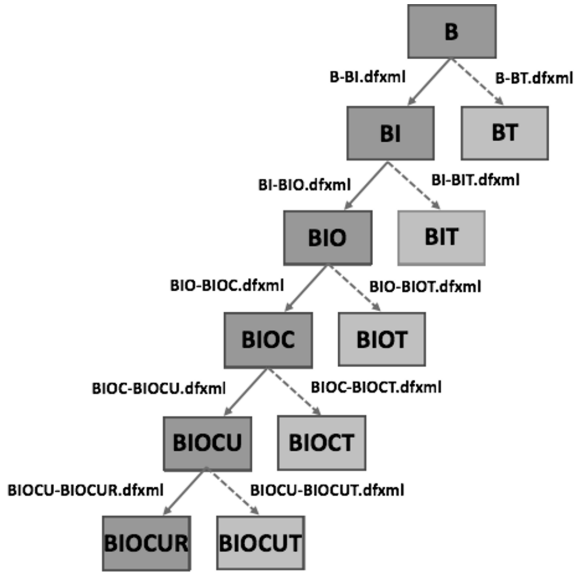

Figure 3. Comparing disk images

We remove the operating-system related activities from "B- BI.dfxml" by comparing "B-BI.dfxml" with "B-BT.dfxml", a dataset that was created from running the base VM instance ("B.vmware") undisturbed during the same time that the other base VM instance was having the application software program installed in it. Upon comparing "B-BI.dfxml" to "B- BT.dfxml", it was observed that there are some file system changes common to both data sets. Therefore it can be assumed that the file system changes common to "B-BI.dfxml" and "B-BT.dfxml" are due to operating system or other application activities in both "do-nothing" and "application- run" paths in this portion of the experiment. These common file system changes are not application artifacts and so are removed from "B-BI.dfxml". Following a similar pattern as described in the above paragraph:

1. "BI-BIO.dfxml" is compared to "BI-BIT.dfxml" to remove common operating system related activities observed during the "Open" phase. This results in reduced "BI- BIO.dfxml".

2. "BIO-BIOC.dfxml" is compared to "BIO-BIOT.dfxml" to remove common operating system related activities observed during the "Close" phase. This results in reduced "BIO-BIOC.dfxml".

3. "BIOC-BIOCU.dfxml" is compared to "BIOC-BIOCT.dfxml" to remove common operating system related activities observed during the "Uninstall" phase. This results in reduced "BIOC-BIOCU.dfxml".

4. "BIOCU-BIOCUR.dfxml" is compared to "BIOCU- BIOCUT.dfxml" to remove common operating system related activities observed during the "Restart" phase, resulting in a reduced "BIOCU-BIOCUR.dfxml".

These comparisons ultimately result in the reduced "application-run" DFXML files.

After eliminating non-application-related file system changes from the DFXML files, we are left with file system activities that are assumed to be attributable to the application's run in the VM environment. The file system activities include file creations, file deletions, file renaming/moving and other file changes.

Upon reviewing the DFXML files, it was observed that the application related files were recorded several times among the different DFXML files under different categories of file system activities. For example, during the application run process for Firefox, configuration and usage files are created during the "Install" phase and modified during the "Run" phase. Due to these repetitions, it was decided during the design stage of this research effort that only file creations would be considered for inclusion in the catalog. Another justification for this design choice is also due to the fact that there are file system changes that remains in the reduced DFXML files that would be not be unique to the specific application. For example, Windows event logs are modified when an application is installed and used. However, it is not wise to include the Windows event logs in the catalog as most application activity results in modification of Windows event logs. If only newly created files are considered for inclusion in the catalog, files such as windows event logs would not be included in the catalog. Therefore, only information about file creations are extracted from the DFXML files. Information about folder creations are not included for ingestion into the catalog because folders have a sub-sector footprint and so would not be useful in our design.

After new file lists are extracted from the DFXML files, the next step is to generate sector hashes of the indicated files. This information is not available in the DFXML files. The DFXML files only contain file metadata such as file inode number, file path, filename, partition, file id, name type, file size, mac time, file system offset, image offset, byte run length, MD5 file hash and SHA1 file hash. Since the goal of this research effort is to be able to accurately infer previously uninstalled applications from partially overwritten files, the catalog needs to be built with sector information. A sector is the smallest physical storage unit on a disk and is typically 512 or 4096 bytes in size. Therefore, in order to build the catalog with sector information from the files attributed to the application of interest, one would need to locate the files in the raw disk images and compute the block hash in increments of 512 or 4096 bytes (we chose 512 in order to be applicable to either 512 byte or 4096 byte images). The files, once located in the raw disk images, are broken down into blocks of 512-byte size, an md5 hash is computed, and the resulting block hash is stored in a JSON file. A custom Python script is used for generating the JSON file containing sector and file information related to each application program. The custom Python script, named "get_sector_hashes.py"1 is used to generate a JSON file that contains sector and file information belonging to the "Firefox19-W7x64" application diskprint in the following example: $ python get_sector_hashes.py {$path to DFXMLs}$ {$path to IMGs$}$ {$path to JSONs}$ Firefox19-W7x64.json Firefox19-W7x64

3.5 Create catalog hashdb and ingest tagged application metadata into it

Once the JSON file containing the sector and file information is created, the next step is to create the catalog database into which the information in the JSON file is ingested. We chose to use the hashdb tool [22] version 3.1.0 for the database since it was specifically designed for hash value storage and lookup. The hashdb tool is also used to create hash databases, import block hashes, scan, and manage block hash databases.

The process of building the catalog hash database involves creating an empty hash database using the hashdb "create" command and ingesting the previously generated JSON file into it using the hashdb "import" command as follows, for "Chrome28-W7x64" application diskprint: $ hashdb create -b 512 Chrome28-W7x64.hdb $ hashdb import Chrome28-W7x64.hdb Chrome28-W7x64.json The content of the created hash database for each application diskprint can be reduced further by removing from it data about blocks that can found in other non-application- related environments, like in a computer system with only the base operating system running. Eliminating these common blocks reduces the false positive matches when the catalog hash database is used to scan against a test image. Removing the common sector information from the application hash database is accomplished using the hashdb "subtract_hash" command. This is accomplished by first creating a new hash database using the hashdb "create" command, ingesting sector and file information collected from a clean Windows 7 64-bit OS image ("W7x64.img") into the database using the hashdb "ingest" command, and then using the hashdb "subtract_hash" to remove sector information that this new database has in common with the application diskprint hash database. This series of commands is run as follows for "Chrome28-W7x64" application diskprint: $ hashdb create -b 512 W7x64.hdb

$ hashdb ingest -r W7x64 W7x64.hdb W7x64.img

$ hashdb subtract_hash Chrome28-W7x64.hdb W7x64.hdb Chrome28-noW7x64.hdb

The next step is to combine all the separate application hash databases into a single hash database. This is accomplished using the hashdb "add_multiple" command. All twenty-nine (29) hash databases are combined as follows:

$ hashdb add_multiple Chrome28-noW7x64.hdb Chrome28-W7x32.hdb ... catalog.hdb

3.6 Scan test images and generate JSON file containing matched sector information

Once the catalog hash database has been built, the next step is to test it for its ability to infer previously uninstalled applications in test disk images. In this research effort, the true test of the improvement of our research methodology over the one previously proposed is to see an increase in the sectors captured per application, and an ability to infer previously uninstalled applications in test images despite significant decay of the application's file artifacts.

The process of scanning test images against the catalog hash database involve using the hashdb "scan_media" command,[1] generates information about matched blocks in a JSON file. The hashdb "scan_media" command is used as follows: $ hashdb scan_media -j e catalog.hdb test.img $>$ catalog-test.match.json

[1] https://github.com/seunfuta/AppDetective/blob/master/get

3.7 Analyze the JSON file containing matched sector information

The information about matched blocks, stored in a JSON file, is then processed to obtain information that demonstrates how much of a previously uninstalled application can be inferred in the scanned test raw disk images. A custom python script was developed to process the JSON file generated in the previous step. The python script, named "process_matched_json.py"2 is used as follows: $ python process_matched_json.py catalog- test.match.json The output of the Python command line execution is a table that shows information about:

- the number of sectors found per application
- the total number of sectors in the catalog per application
- the sector percentage, which is number of found sectors/total number of sectors per application
- the weighted sector percentage, which considers the frequency of each matched sector among the 29 applications in the catalog
- the number of files, based on the sectors found, per application
- the total number of files in the catalog per application
- the file percentage, which is the number of files with found sectors/total number of files per application
- the weighted file percentage, which considers the fraction matched sector/total sectors for each matched file per application

The final step in our research methodology is to compare the tables generated in our research work to the tables generated based on the work of Jones. The tables are generated with the two catalogs scanned against identical test images. Identifying more sectors per applications known to be previously installed in the test images would be sufficient as the initial proof of concept of our hypothesis.

4. FINDINGS AND ANALYSIS

This experiment was conducted with 16 applications in 3 operating systems environments, namely Windows XP, Windows 7 32-bit and Windows 7 64-bit (not every application was printed for every OS). Table 1 shows

App-OS	WinXP	Win7x32	Win7x64
Advanced Keylogger	X		
Chrome28	X	X	X
Eraser		X	
Firefox	X	X	X
HxD Hex Editor		X	
Invisible Secrets	X		
MS Office	X	X	X
Python264	X		
Safari517	X	X	X
Sdelete		X	X
Thunderbird2	X		
TrueCrypt63	X		
UPX		X	X
WinRar5beta		X	X
WinZip17Pro		X	X
Wireshark		X	X

Table 1. NIST Diskprints

the distribution of the application diskprints created, which were the same applications and operating systems used in the NIST Diskprinting effort [23]. The combination of the applications and operating systems resulted into 29 application diskprints that were used to build the catalog hash database.

A diskprint represents a set of sequential VM snapshots, each snapshot capturing a slice of time in the software's life cycle on the system.

The experiment, as presumed, revealed that there were file system changes that were common between disk images obtained from the "application-run" path of the experiment and those obtained from the "do-nothing" path. Eliminating these common file system changes reduced the number of files whose sectors are included in the catalog when compared to all file differencing files and sectors, but resulted in more files and sectors than the previous method after white list application. A review of the remaining files showed that they are unique to the "application-run" branch and thus true indicators of activities that are attributable to the application of interest. These includes created files that can be directly linked to the application installation package content and other related file systems changes. All these are good candidates for consideration for the catalog hash database building.

Table 2 Shows the total sector hashes and files per diskprints in our catalog and in the work of Jones.

The higher sector and file count per application diskprint in our method supports our assertion that this is a more complete catalog. Manual review of the artifact source files indicates that these are application related files, hence the new catalog has maintained accuracy as well.

Jones used the following equations to compute a score based on matched sectors:

sector %DP = sector_matches / sectors_totalDP

file %DP = files_found / files_totalDP

We tested our catalog using the same twelve (12) test images used by Jones and these scoring equations; the full results are presented in Tables 4 and 5 (in the Appendix). Our catalog typically scored lower than the Jones catalog due to the fact that the Jones catalog is smaller. Since total sectors (and total files) are a denominator in Jones' scoring equations, our scores are usually lower even when we match more sectors. Given a more complete and still accurate catalog (no irrelevant artifacts), a better measure would emphasize the number of sectors (and files) matched, rather than the % of total sectors or files. Further, our more complete catalog is inherently more decay tolerant, as it provides more sectors against which matches can be made. This revised scoring equation is noted as future work, but here we present a comparison of the number of sectors and files matched (Table 3) [2]

$$\text{weighted file } \%_{DP} = \left(\sum_{F=1}^{num\_file\_matches} \frac{matched_s ectorsF}{total_s coresF} \right) / files\_total_{DP}$$

$$\text{weighted sector} \%_{DP} = \left(\sum_{S=1}^{num\_sector\_matches} \frac{1}{fre1_s} \right) / sectors\_total_{DP}$$

4.1 Result of Catalog Scan against Test Images with Single Applications

The same five single-application test images used by Jones for testing their catalog hash database were used to test our newly built catalog hash database. Testing with the same test images would allow one to compare the two catalog hash databases and methodologies for difference in accuracy of detection. The five single-application test images were built using Chrome28, Firefox19, UPX, Winrar5beta, and sdelete within Windows 7 64-bit environments. The results of the scans, using the five images which contained the installation, use and uninstallation of a single

[2] https://github.com/seunfuta/AppDetective/blob/master/process_matched_json.p

	Current		Jones et al	
diskprint	sectors	files	sectors	files
AdvKeylogger-WinXP	34,021	113	4,682	25
Chrome28-W7x32	1,628,813	417	563,047	181
Chrome28-W7x64	1,326,264	394	618,344	191
Chrome28-WinXP	1,302,446	349	616,707	175
eraser-W7x32	473,414	106	88,952	32
Firefox19-W7x32	772,936	285	136,069	79
Firefox19-W7x64	327,374	235	143,607	99
Firefox19-WinXP	306,791	203	136,276	93
HxD171-W7x32	16,003	35	8,005	18
InvSecrets21-WinXP	24,593	111	1,197	18
OfficePro2003-W7x32	3,655,215	6,322	1,435,417	2,961
OfficePro2003-W7x64	2,735,821	1,834	1,073,253	864
OfficePro2003-WinXP	4,707,778	9,255	1,415,905	2,967
Python264-WinXP	110,556	4,427	47,062	2,035
Safari157-W7x32	1,068,187	2,975	407,531	1,434
Safari157-W7x64	847,899	1,736	301,256	806
Safari157-WinXP	1,015,377	2,252	312,454	804
sdelete-W7x32	1,740	7	249	3
sdelete-W7x64	4,054	6	225	2
Thunderbird2-WinXP	123,855	335	44,669	153
TrueCrypt63-WinXP	21,750	41	9,014	15
UPX-W7x32	3,324	17	1,340	8
UPX-W7x64	2,260	19	798	8
Winrar5beta-W7x32	641,637	135	9,209	41
Winrar5beta-W7x64	36,947	107	15,311	62
Winzip17pro-W7x32	1,449,498	355	323,735	149
Winzip17pro-W7x64	750,663	320	283,153	149
Wireshark-W7x32	681,631	505	102,309	237
Wireshark-W7x64	448,605	475	129,684	223
Total	24,519,452	33,371	8,229,460	13,832

Table 2. Total hashes and files per Application Diskprint

application, shows improvement in the number of sectors matched for known previously uninstalled applications. Table 3 summarizes the results and Table 4 shows the full results of the comparisons.

4.2 Result of Catalog Scan against Test Images with Multiple Applications

Similarly, the same three multiple-application test images used by Jones for testing their catalog hash database were used to test our newly built catalog hash database. Each test image contained the installation, use and uninstallation of multiple applications (Chrome28-Firefox19, Chrome28- Firefox19-Safari517 and Winrar5beta-Winzip17pro) in Windows 7 64-bit environments. The test results show improvement in the number of sectors matched for known previously uninstalled applications. Table 3 summarizes the results and Table 5 shows the full results of the comparisons.

4.3 Result of Catalog Scan against M57 Dataset

The new catalog hash database was also tested against the M57 Patents dataset [3]. The M57-Patents dataset corresponds to a case involving four employees of a fictitious corporation, three of whom were involved in various types of criminal activity. In producing the dataset, the scenario participants engaged in scripted and normal user activities every day for one month. Researchers made forensic images of the user workstations at the end of each day. Testing the catalog hash

	Diff		Diff %	
diskprint	sectors	files	sectors	files
AdvKeylogger-WinXP	29,339	88	627%	352%
Chrome28-W7x32	1,065,766	236	189%	130%
Chrome28-W7x64	707,920	203	114%	106%
Chrome28-WinXP	685,739	174	111%	99%
eraser-W7x32	384,462	74	432%	231%
Firefox19-W7x32	636,867	206	468%	261%
Firefox19-W7x64	183,767	136	128%	137%
Firefox19-WinXP	170,515	110	125%	118%
HxD171-W7x32	7,998	17	100%	94%
InvSecrets21-WinXP	23,396	93	1955%	517%
OfficePro2003-W7x32	2,219,798	3,361	155%	114%
OfficePro2003-W7x64	1,662,568	970	155%	112%
OfficePro2003-WinXP	3,291,873	6,288	232%	212%
Python264-WinXP	63,494	2,392	135%	118%
Safari157-W7x32	660,656	1,541	162%	107%
Safari157-W7x64	546,643	930	181%	115%
Safari157-WinXP	702,923	1,448	225%	180%
sdelete-W7x32	1,491	4	599%	133%
sdelete-W7x64	3,829	4	1702%	200%
Thunderbird2-WinXP	79,186	182	177%	119%
TrueCrypt63-WinXP	12,736	26	141%	173%
UPX-W7x32	1,984	9	148%	113%
UPX-W7x64	1,462	11	183%	138%
Winrar5beta-W7x32	632,428	94	6867%	229%
Winrar5beta-W7x64	21,636	45	141%	73%
Winzip17pro-W7x32	1,125,763	206	348%	138%
Winzip17pro-W7x64	467,510	171	165%	115%
Wireshark-W7x32	579,322	268	566%	113%
Wireshark-W7x64	318,921	252	246%	113%

Table 3. Sector and File Match Comparison between Methods

database using these images would help determine the ability of our catalog hash dataset to accurately infer the presence of previously uninstalled applications under semi-realistic conditions. Testing against the M57-Patents dataset also completes our comparison to the methodology and catalog hash database of Jones.

Chrome28, Firefox19-W7x64												
	Current						Jones et al					
diskprintName	sectors	sector%	w_sector%	files	file%	w_file%	sectors	sector%	w_sector%	files	file%	w_file%
Chrome28-W7x32	876,383	53.81%	15.97%	222	53.24%	22.73%	375,906	66.76%	22.12%	102	56.35%	26.04%
Chrome28-W7x64	911,897	68.76%	20.36%	230	58.38%	26.04%	441,175	71.35%	25.42%	113	59.16%	28.25%
Chrome28-WinXP	910,660	69.92%	20.70%	198	56.73%	25.17%	441,107	71.53%	25.48%	99	56.57%	26.88%
Firefox19-W7x32	416,080	53.83%	15.89%	220	77.19%	61.25%	126,693	93.11%	30.45%	66	83.54%	76.00%
Firefox19-W7x64	289,043	88.29%	28.75%	198	84.26%	67.27%	132,776	92.46%	29.89%	85	85.86%	74.26%
Firefox19-WinXP	272,363	88.78%	29.08%	164	80.79%	67.57%	127,284	93.40%	30.70%	76	81.72%	73.56%
Chrome28, Firefox19, Safari517-W7x64												
	Current						Jones et al					
diskprintName	sectors	sector%	w_sector%	files	file%	w_file%	sectors	sector%	w_sector%	files	file%	w_file%
Chrome28-W7x32	242,532	14.89%	3.18%	207	49.64%	15.63%	51,827	9.20%	3.07%	94	51.93%	17.56%
Chrome28-W7x64	254,361	19.18%	4.08%	216	54.82%	18.07%	117,143	18.94%	8.08%	105	54.97%	19.77%
Chrome28-WinXP	254,186	19.52%	4.15%	188	53.87%	16.70%	117,077	18.98%	8.09%	93	53.14%	18.16%
Firefox19-W7x32	77,977	10.09%	1.61%	138	48.42%	4.71%	4,119	3.03%	1.01%	40	50.63%	3.45%
Firefox19-W7x64	10,014	3.06%	0.99%	123	52.34%	4.45%	4,297	2.99%	1.01%	48	48.48%	4.00%
Firefox19-WinXP	9,646	3.14%	1.01%	106	52.22%	4.14%	4,260	3.13%	1.06%	44	47.31%	3.75%
Safari157-W7x32	923,854	86.49%	28.83%	2,608	87.66%	81.54%	349,176	85.68%	28.55%	1,252	87.31%	82.02%
Safari157-W7x64	732,700	86.41%	28.94%	1,521	87.62%	81.73%	255,830	84.92%	28.30%	708	87.84%	82.48%
Safari157-WinXP	886,689	87.33%	29.11%	1,980	87.92%	83.12%	266,329	85.24%	28.41%	703	87.44%	82.24%
Winrar5beta, Winzip17pro-W7x64												
	Current						Jones et al					
diskprintName	sectors	sector%	w_sector%	files	file%	w_file%	sectors	sector%	w_sector%	files	file%	w_file%
Winrar5beta-W7x32	4,514	0.70%	0.32%	53	39.26%	3.75%	1,993	21.64%	10.80%	23	56.10%	6.74%
Winrar5beta-W7x64	6,046	16.36%	8.05%	54	50.47%	6.46%	3,908	25.52%	12.75%	29	46.77%	6.66%
Winzip17pro-W7x32	1,104,163	76.18%	38.08%	315	88.73%	85.16%	302,247	93.36%	49.58%	141	94.63%	93.94%
Winzip17pro-W7x64	725,194	96.61%	48.36%	308	96.25%	93.34%	274,260	96.86%	48.52%	146	97.99%	96.49%

Table 4. Multiple Application Test Case Results

CHARLIE	Current		Jones et al		JO	Current		Jones et al	
diskprintName	sectors	files	sectors	files	diskprintName	sectors	files	sectors	files
Python264-WinXP	101,164	4,381	46,289	2,025	Python264-WinXP	101,167	4,382	46,289	2,025
Thunderbird2-WinXP	16,848	281	8,033	129	TrueCrypt63-WinXP	17,802	15	8,680	8
InvSecrets21-WinXP	5,096	41	1,173	12	Thunderbird2-WinXP	537	141	236	63
Safari157-W7x64	8,001	328	2,867	127	Safari157-W7x64	7,980	332	2,857	131
Safari157-WinXP	8,549	413	2,912	128	Safari157-WinXP	8,507	419	2,902	132
Safari157-W7x32	8,468	434	3,125	177	Safari157-W7x32	8,471	442	3,126	182
AdvKeylogger-WinXP	124	11	1	1	UPX-W7x32	2	2	1	1
eraser-W7x32	157	13	31	3	UPX-W7x64	2	2	1	1
Firefox19-WinXP	338	65	288	25	AdvKeylogger-WinXP	265	13	1	1
Firefox19-W7x64	983	77	290	26	InvSecrets21-WinXP	132	15	0	0

PAT	Current		Jones et al		TERRY	Current		Jones et al	
diskprintName	sectors	files	sectors	files	diskprintName	sectors	files	sectors	files
Python264-WinXP	101,161	4,382	46,289	2,025	Python264-WinXP	73,755	3,958	33,091	1,814
AdvKeylogger-WinXP	308	27	20	7	HxD171-W7x32	5,778	10	2,862	4
HxD171-W7x32	5,398	6	2,698	3	Thunderbird2-WinXP	684	157	303	71
Thunderbird2-WinXP	520	137	224	61	Winzip17pro-W7x64	13,209	112	6,711	53
InvSecrets21-WinXP	124	17	0	0	Winzip17pro-W7x32	13,932	121	6,724	54
Chrome28-W7x64	513	43	258	21	Safari157-W7x32	1,728	290	718	130
eraser-W7x32	214	12	48	3	Safari157-W7x64	1,264	178	498	76
Firefox19-WinXP	1,045	107	361	49	eraser-W7x32	231	15	62	3
Firefox19-W7x64	1,187	123	364	51	Firefox19-WinXP	1,113	71	390	26
Firefox19-W7x32	1,376	127	341	41	Safari157-WinXP	1,358	203	512	77

Table 5. M57 Patent Scenario Results

The test images used are the final day snapshots of the workstations of the four employees, namely Charlie, Jo, Pat and Terry. The results from scanning the final day images for the four scenario users are summarized in Table 3.

A review of Tables 3-5 shows that our methodology has resulted in more identified sectors and files per application known to be previously present on the test images. However, the previous approach of inferring application presence based on sector percentages, weighted sector percentages, file percentages and weighted file percentages does not take advantage of the more complete catalog, nor does it capture the increased decay tolerance of the new catalog. To demonstrate this point, we used our catalog to scan all of the M57 Pat workstation images collected over the one-month period and plotted the count of sectors associated with Advanced Keylogger over the time period of the scenario. Advanced Keylogger is malware that was installed and active 12/3 and 12/4, then uninstalled. Figure 4 shows how our approach was able to identify considerably more sectors (308 vs 20 for the final image captured) associated with the Advanced Keylogger application program in Pat's system, even after uninstallation and continued system use. In digital forensic investigations, we frequently analyze systems well after the event has occurred. The presence of significantly more sectors against which we can match may be

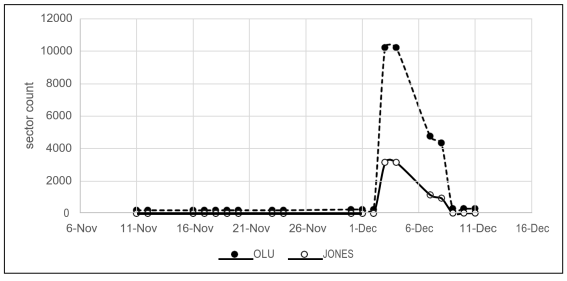

Figure 4. Sector artifact persistence for Advanced Keylogger on Pat's M57 system

the difference between whether we identify a previoulsy uninstalled application or not.

5. CONCLUSION

In this work, we proposed a method of generating an application-related artifact set for our catalog hash database by extending the sequential snapshot file differencing method of Jones, adding a "do-nothing" branch to the collection activity. This additional process eliminates artifacts due to operating system and other application activities while retaining more of the relevant artifacts than the previous method. This provides a more decay tolerant catalog, although we note the need for a new quantitative measure of application presence which takes advantage of the more complete catalog.

6. FUTURE WORK

Future work involves reviewing the makeup of the catalog hash database to see how sectors that don't contribute much value to the application inference calculation can be identified and eliminated. In addition, a revised measure of application presence and thresholds using our more complete catalog needs to be devised. Such a measure may incorporate factors such as the application footprint, relative sector locations, artifact decay contributing factors, etc. We also intend to generate more test images to continue testing our catalog hash database.

REFERENCES

[1] J. Haggerty and M. Taylor, "Forsigs: Forensic signature analysis of the hard drive for multimedia file fingerprints," *New Approaches Secur. Priv. Trust Complex Environ.*, pp. 1–12, 2007.

[2] J. Jones, T. Khan, K. Laskey, A. Nelson, M. Laamanen, and

[3] White, "Inferring previously uninstalled applications from digital traces," in *Proceedings of the Conference on Digital Forensics, Security and Law*, 2016, pp. 113–130.

[4] K. Woods, C. A. Lee, S. Garfinkel, D. Dittrich, A. Russell, and K. Kearton, "Creating realistic corpora for security and forensic education," in *Proceedings of the Conference on Digital Forensics, Security and Law*, 2011, p. 123.

[5] D. Quick and K.-K. R. Choo, "Dropbox analysis: Data remnants on user machines," *Digit. Investig.*, vol. 10, no. 1, pp. 3–18, 2013.

[6] M. Geiger and L. F. Cranor, "Scrubbing stubborn data: An evaluation of counter-forensic privacy tools," *IEEE Secur. Priv.*, vol. 4, no. 5, pp. 16–25, 2006.

[7] A. Margosis and M. E. Russinovich, *Windows Sysinternals administrator's reference*. Pearson Education, 2011.

[8] S. Garfinkel, A. J. Nelson, and J. Young, "A general strategy for differential forensic analysis," *Digit. Investig.*, vol. 9, Supplement, pp. S50–S59, Aug. 2012.

[9] CGSecurity, "TestDisk - Partition Recovery and File Undelete," 04-Jun-2016. [Online]. Available: https://www.cgsecurity.org/wiki/TestDisk. [Accessed: 22-Dec-2018].

[10] A. Ravi, T. R. Kumar, and A. R. Mathew, "A method for carving fragmented document and image files," in *Advances in Human Machine Interaction (HMI), 2016 International Conference on*, 2016, pp. 1–6.

[11] G. Richard, *Scalpel*. The Sleuth Kit, 2005.

[12] J. H. Jones and T. M. Khan, "A method and implementation for the empirical study of deleted file persistence in digital devices and media," in *Computing and Communication Workshop and Conference (CCWC), 2017 IEEE 7th Annual*, 2017, pp. 1–7.

[13] M. H. Ligh, A. Case, J. Levy, and A. Walters, *The art of memory forensics: detecting malware and threats in windows, linux, and Mac memory*. John Wiley & Sons, 2014.

[14] A. Nelson, "XML Conversion of the Windows Registry for Forensic Processing and Distribution," in *Advances in Digital Forensics VIII: 8th IFIP WG 11.9 International Conference on Digital Forensics, Pretoria, South Africa, January 3-5, 2012, Revised Selected Papers*, G. Peterson and S. Shenoi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 51–65.

[15] E. Casey, "Network traffic as a source of evidence: tool strengths, weaknesses, and future needs," *Digit. Investig.*, vol. 1, no. 1, pp. 28–43, 2004.

[16] L. Garber, "Encase: A case study in computer-forensic technology," *IEEE Comput. Mag. January*, 2001.

[17] E. Casey, *Digital evidence and computer crime: Forensic science, computers, and the internet*. Academic press, 2011.

[18] C. Painter, "Threats to the Net: Trends and Law Enforcement Responses," in *Crime and Technology: New Frontiers for Regulation, Law Enforcement and Research*, E. U. Savona, Ed. Dordrecht: Springer Netherlands, 2004, pp. 69–77.

[19] G. G. Richard III and V. Roussev, "Scalpel: A Frugal, High Performance File Carver.," in *DFRWS*, 2005.

[20] C. J. Veenman, "Statistical Disk Cluster Classification for File Carving," in *Third International Symposium on Information Assurance and Security*, 2007, pp. 393–398.

[21] S. Garfinkel, A. Nelson, D. White, and V. Roussev, "Using purpose-built functions and block hashes to enable small block and sub-file forensics," *Digit. Investig.*, vol. 7, pp. S13–S23, 2010.

[22] S. L. Garfinkel and M. McCarrin, "Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb," *Digit. Investig.*, vol. 14, pp. S95–S105, 2015.

[23] M. Laamanen and A. Nelson, *NSRL Next Generation- Diskprinting. Forensics@ NIST, Gaithersburg, MD, December 3, 2014. Last accessed 10.4. 15*. 2014.

