FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

# Detecting Privacy Leaks in the Private Browsing Mode of Modern Web Browsers Through Process Monitoring

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Software Engineering & Internet Computing**

by

**Herbert Brunner, BSc**
Registration Number 0627669

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl
Assistance: Univ.Lektor Dipl.-Ing. Martin Mulazzani, PhD

Vienna, 21. August 2014 ____________________ ____________________
(Signature of Author) (Signature of Advisor)

# Erklärung zur Verfassung der Arbeit

Herbert Brunner, BSc
Gartengasse 17 / 2, 1050, Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

| | |
|---|---|
| (Ort, Datum) | (Unterschrift Verfasser) |

# Abstract

A main topic regarding modern web browsers is user privacy. When surfing on the Internet, web browsers typically store user-related browsing data, such as cookies, browsing history and web site banners, on the local computer system. Saving that data locally may pose a security risk to Internet users, as it can be recovered from a computer's hard disk by means of various forensic tools. To tackle this privacy issue, web browser vendors introduced the private browsing mode, which promises not to store sensitive user data to the local system, in order to preserve user privacy. As there exists no guarantee that the private browsing mode of modern web browsers has been implemented and tested thoroughly from a forensically standpoint of view, in this work a proof-of-concept is provided which examines this mode by means of a new forensic analysis approach.

This approach takes advantage of two frameworks. One framework has been used for performing automated web browser tests, whereas the other one has been implemented for forensic analysis purposes. The key feature of the analysis framework is based on the concept of process monitoring, which offers the possibility to log file system events that have been induced by a web browser during a private browsing session. The collected file system event log files have been used for file recovery purposes in conjunction with Digital Forensics XML (DFMXL) files. Generally, DFXMLs provide digital examiners with valuable information about file objects (e.g. last access time, file size, allocation status).

The experimental evaluation of this work is based upon this information, in order to retrieve those Internet artefacts which have been accessed when surfing the Internet in private mode.

The evaluation of the results has shown that the private modes of the tested web browsers have been implemented differently. The amount of recovered Internet artefacts has varied depending on the web sites as well as on the web browsers that have been used for testing. From forensically standpoint of view, it has been found that private browsing artefacts can be recovered effectively by using process monitor log files.

# Kurzfassung

Im Laufe der letzten Jahre hat der Begriff Privatsphäre für Internet-Benutzer immer mehr an Bedeutung gewonnen. Normalerweise legen Web-Browser beim Internet-Surfen verschiedene Artefakte auf der Festplatte eines Rechner ab, um beispielsweise bessere Antwortzeiten in Bezug auf Benutzeranfragen zu erzielen. Allerdings birgt dieses Default-Verhalten von Web-Browsern einige Sicherheitsrisiken für deren Benutzer mit sich. Es können zum Beispiel Internet-Artefakte, wie Bild- und Textdateien, von der Festplatte eines Rechners rekonstruiert werden, nachdem der Web-Browser bereits beendet wurde.
Entsprechend haben die Hersteller von modernen Web-Browsern reagiert und ihre Produkte mit einem Feature ausgestattet, das die Privatsphäre von Internet-Benutzern schützen soll. Dieses Features wird privater Modus genannt. Wenn ein Web-Browser im privaten Modus gestartet wird, lässt er üblicherweise keine digitalen Spuren auf der Festplatte zurück, die über das Surf-Verhalten eines Internet-Benutzers Aufschluss geben. Nachdem allerdings keine Garantie besteht, dass der private Modus aus forensischer Sicht fehlerfrei seitens der Web-Browser Hersteller implementiert wurde, bedarf es eines neuen Ansatzes, um die Qualität der implementierten Software hinsichtlich der Privatsphäre von Benutzern zu prüfen. Im Rahmen dieser Arbeit wurde eine neue Methode entwickelt, um den privaten Modus von modernen Web-Browsern aus forensischer Sicht zu überprüfen.
Der praktische Teil dieser Arbeit umfasst ein Test- und ein Analyse-Framework. Das Test-Framework wird für das automatisierte Testen von ausgewählten Web-Browsern verwendet. Das Analyse-Framework wird zum Auswerten der Testergebnisse eingesetzt. Das Kernkonzept dieser Arbeit, welches in beiden Frameworks umgesetzt wurde, basiert auf der Überwachung von Prozessen. Beim Testen von Web-Browsern werden hierfür Log-Dateien erstellt, welche jene Dateisystem-Events enthalten, die der private Modus eines Web-Browsers am lokalen Rechner während des Internet-Surfens generiert hat. Diese Log-Dateien werden anschließend vom Analyse-Framework dazu verwendet, um einerseits zu bestimmen, wann ein Web-Browser im privaten Modus Dateien auf die Festplatte schreibt. Andererseits werden sie dazu verwendet, um Internet-Artefakte von der Festplatte zu bergen, um in weiterer Folge bestimmen zu können, welche Änderungen bei einzelnen Dateien (z.B. beim Dateiinhalt) stattgefunden haben. Hierfür werden Digital Forensics XML (DFXML) Dateien eingesetzt.
Die Evaluation hat ergeben, dass die getesteten Web-Browser den privaten Modus in unterschiedlicher Qualität implementieren. In Abhängigkeit von den getesteten Web-Seiten und vom verwendeten Web-Browser haben sich Unterschiede in der Anzahl der rekonstruierten Internet-Artefakte gezeigt. Die Genauigkeit der eingesetzten, forensischen Methode hat sich positiv auf das Finden von digitalen Spuren hinsichtlich des privaten Modus erwiesen.

# Contents

# List of Figures

# List of Listings

# List of Tables

CHAPTER 1

# Introduction

## 1.1 Problem Definition

Within the last two decades the Internet has rapidly expanded. More and more people frequently use the Internet - with suitable software like web browsers - in business or for private purposes (e.g. email, instant messaging).
Web browsers provide many features for retrieving, traversing and displaying various content from the Internet. Several web browser products are available on the market, which differentiate not only in layout, but in handling or the individual implementation.
When surfing the Internet modern web browsers are executed in public browsing mode by default. This basically means that user-related browsing data, such as cookies or bookmarks, are stored on a computer's hard drive. By collecting, storing and processing these data web browsers can increase their performance to be able to handle user requests quickly.
In spite of the higher throughput of web browsers, capturing and storing individual browsing data can violate user privacy. For example, if a computer is hacked by a local intruder, the perpetrator will gain access to individual browsing data. Moreover, the intruder will be able to draw conclusions about the browsing activities of the users of that computer. Depending on the value of the individual browsing data (e.g. passwords, credit card numbers), an attacker could recover and steal that data. To tackle this threat, modern web browsers offer a ”private browsing mode” [ [87], Sammons, 2012 ], which claims not to store any user-related browsing data to the local hard disk, in order to preserve user privacy. Due to the variety of modern web browsers and their rapid development, private browsing modes are expected to work differently (depending on the software vendor). As a consequence, the privacy of local users mainly depends on a properly functioning private mode when dealing with web browsers.
From a forensically standpoint of view there exists no guarantee that the private browsing mode of modern web browsers has been implemented and tested sufficiently. Therefore, digital investigation needs to be done, in order to detect privacy leaks in the private mode of modern web browsers.

## 1.2 Aim of This Thesis

The main goal of this master's thesis is to provide a forensic approach for investigating the private browsing mode of modern web browser. A proof-of-concept is implemented by providing a framework for web browser testing combined with a framework for the forensic analysis of web browsing data. This approach can be used to determine whether or not user-related browsing data has been stored on the local file system in the context of a private browsing session. Furthermore, this approach can be used to recover user-related browsing data from the local system. As modern web browsers can be used in private mode manually or automatically (e.g. with a framework), the main research question to answer is whether there are any differences from a forensically standpoint of view when a web browser is used in private mode manually, in contrast to when it is used in private mode automatically.
Assuming that there are differences between executing the private mode manually and automatically, this thesis aims to evaluate these differences and give recommendations for the practical use of the web browsers that have been tested.

## 1.3 Structure of This Thesis

In chapter 2, the field of digital forensics is introduced. An introduction to file systems is presented. Additionally, common forensic analysis techniques are described. Digital anti-forensic techniques are presented and an explanation is provided of how this topic is related to the private browsing mode of web browsers. Chapter 3 illustrates the new approach for investigating the private mode of modern web browsers. The field of virtualisation is introduced as well as process monitoring. In chapter 4, the implementation of the testing framework as well as of the forensic analysis framework is described. Chapter 5 defines test scenarios that have been evaluated. Additionally, the findings are reported. In chapter 6, the test results are discussed. In addition the chapter discusses the strength and limitations of the proposed approach. Chapter 7 concludes this master's thesis.

CHAPTER 2

# Background

## 2.1 Introduction to Digital Forensics

Within the last decades computers and networks have grown rapidly and according to this fact computer crime did as well. In [21] computer crime is also referred to as cyber crime being only one related term out of many. In cases of cyber crimes the computer, the network or a digital media must be a central part of the criminal act, contrary to crimes that involve an unforeseen computer usage, e.g. during an incident. Crimes like terrorism, child pornography or drug dealing may involve some kind of digital evidence (e.g. documents, images or movies), which is located on digital media like hard drives or compact disks. Therefore, collecting digital evidence within the context of a digital investigation will become more important in order to detect and prevent such crimes [21,37,38].
Digital investigation is not limited to cyber crime, but it can be used in different contexts and there are several other explanations of this term available. As stated by Carrier, ”A digital investigation is a process where we develop and test hypotheses that answer questions about digital events.” [ [33], Carrier, 2005 ]. Brown defines digital investigation with the words ”digital forensics” [ [31], Brown, 2009 ] referring to ” ... the art and science of applying computer science knowledge and skills to aid the legal process.” [ [31], Brown, 2009 ]. Another definition of digital forensics is given in [50] which uses the terms ”digital archaeology” [ [50], Farmer and Venema, 2005 ] and ”digital geology” [ [50], Farmer and Venema, 2005 ]. Digital archaeology describes the direct consequences that occur on a system which follow from user actions, e.g. the modification of file timestamps, whereas digital geology means that the outcome of digital processes is not directly affected by a user, e.g. the allocation of file space. In any case, either users or processes trigger events that create digital evidence. Carrier defines digital evidence as ” ... a digital object that contains reliable information that supports or refutes a hypothesis.” [ [33], Carrier, 2005 ], in order words, digital evidence supports or refutes the hypothesis which is about to be tested within the context of a digital investigation.
According to [22,50], a digital investigation is about extracting data of a system and processing it

to gain more information about the truth of events. To achieve this goal the next section presents an overall process for managing digital investigations.

### 2.1.1 The Digital Forensic Process

Various proper definitions of digital forensic process models are available, for example one in [81] and another in [82]. For reasons of simplicity this work covers the digital investiga-

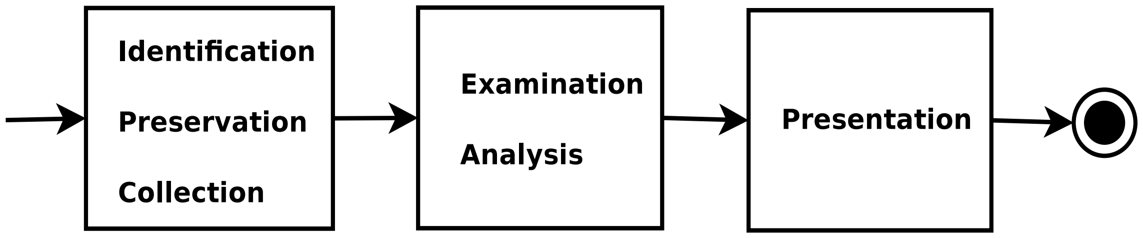


**Figure 2.1:** Schematic representation of the digital forensic process model [37]

tion process model according to [37], as can be seen in figure 2.1. Altheide and Carvey [22] summarise the steps of this process model by describing three consecutive phases [22]:

1. Acquirement

   As part of the initial acquisition process, proper information gathering is a compelling task. The media which contain evidentiary value, such as optical media, mobile phones or disk drives, are identified and collected for further examination. There may be additional artefacts like manuals for a correct device handling. In some cases precise media specific information like disk geometry, hidden partitions or unknown file systems are retrieved [48, 54].
   During the acquisition phase it is recommended to preserve the state of the original media by creating a "working copy" [ [22], Altheide and Carvey, 2011 ]. This copy is a duplicate of the original media and more appropriate for the next investigation step, since forensic tools are known to modify and destroy disk data respectively.
   Additionally, the preservation of digital data focuses on techniques to ensure data integrity, e.g. checksums or cryptographic hashes [31].

2. Examination

   The preserved data is filtered and divided into two sets. One set consists of artefacts that are of no potential value and the other one contains artefacts which are relevant for further examination [31]. These artefacts are subjected to an adequate analysis (e.g. file content analysis) and the gained results are interpreted by the investigator [22]. The outcome of this phase mostly depends on the skills and experience of the investigator, because digital evidence occurs in various forms and may come from different locations being difficult to analyse [44].

3. Reporting

   A proper documentation of all findings has to be managed throughout the whole process of a digital investigation to ensure the completeness of the chain of custody in a way which is admissible in court of law, i.e. the forensic report needs to be documented thoroughly to be able to defend the documented actions against the adverse party [48]. Typically, the evidence found during the investigation is extracted from the analysed media and transferred to portable media like a CD-ROM or a DVD-ROM [22, 31].

Philipp et al. [79] suggest to archive the collected evidence to have it for future use. Basically this means that a bundle is created, which consists of a proper maintained process documentation and the archived data. It is recommend to store such bundles securely to protect it from unauthorised access, e.g. theft, as well as environmental damage, e.g fire or flooding.

### 2.1.2 Disk Imaging Techniques

As part of the digital investigation process, proper data acquisition techniques are necessary when creating a working copy, i.e. the integrity of the original media has to be preserved. Therefore, choosing a forensically sound disk imaging technique may not always be a simple task. In [57] a methodology is presented which allows investigators to formulate requirements for picking an appropriate imaging technique. Based upon different criteria, such as memory type (e.g. physical, virtual) or data destination format (e.g. raw, structured), an imaging technique can be evaluated which fits best.
Data duplication is established in a forensically sound manner by using approved tools that do not alter the original storage device during the duplication process. In general, either a physical or a software write-blocker is used to accomplish a forensic copy of a storage device. What method to use depends on different circumstances. If the storage media cannot be easily removed because it is built-in a mobile device, e.g. a laptop, then a write-blocking software may be used, otherwise a physical write-blocker is recommended. In the latter case the storage media is removed from the original workstation and is connected to a forensic workstation via hardware write-blocker [44].
In agreement with NIST Special Publication 800-86 there are two ways of creating a working copy [64]:

- Logic Copy

  Establishing a simple logic copy of a volume is performed by copying its files and directories [48]. Additional file system information like deleted content (e.g. files, directories), partially overwritten files or free space is usually not part of a logic copy [64,87]. The data contained in a logic copy will depend on the tools that are used to conduct the copy [87].

- Bit Copy

  When performing a bit copy the original source media is copied bit-by-bit to a new destination, i.e. this copy represents the exact duplicate of the original media [48]. Data can be copied by using two different approaches: ”disk-to-disk” [ [64], Kent et al., 2006 ] or ”disk-to-file” [ [64], Kent et al., 2006 ]. The disk-to-disk approach copies the data directly from one media to another one, whereas disk-to-file duplicates a media by copying its contents to a single file. Usually bit copies take longer to perform and more disk storage is required, due to the fact that more details are copied.

Sammons [87] recommends to choose the destination media’s size larger than the source’s one, because when conducting a bit-by-bit copy extra file system data may have to be stored. Before the cloning process can take place, a forensically ”clean” destination media is required, i.e. which does not contain any data. Cleaning can be accomplished by using tools that will completely overwrite the media with ones or zeros [87].

### 2.1.3 The Volatility of Data

During a forensic investigation digital evidence can be collected by acquiring data of a dead system, which is powered off, or a live one that is up and running [87]. The term volatile is applied to data, which can only be collected from live systems, in opposition to non-volatile data which is available on both dead and live systems [61].
Typically, volatile data is kept in RAM of a computer. This data can contain information about open network connections, recently accessed files, login sessions, running processes or password hashes. Non-volatile data is saved permanently on a media, e.g. software configuration files, password files. Although non-volatile data can be acquired from dead or live systems, usually the system is shut down when non-volatile data is collected in the course of a digital investigation. A system shutdown can be accomplished by shutting down the operating system or by pulling the power plug.
Traditional forensic investigations were dealing with the inspection of non-volatile data; volatile data was neglected. Nowadays neglecting volatile data may be insufficient in the case of an investigation, because software products like instant messengers may have file logging disabled by default. Digital evidence regarding such messaging applications may be residing in RAM only. Therefore, forensic examiners may lose valuable information by ignoring the content of the RAM [61,64,87].
Acquiring data from a live system could be one possible solution for this problem taking into account that collecting live data has its drawbacks. A forensically sound approach to preserve digital evidence should not change a system, it is not of significance whether dead or alive. In fact, live acquisition of data modifies the running system which may not be a forensically sound approach in court of law. Whether performing a live acquisition or not, in any case, it is advisable to create an accurate documentation [39].
The following enumeration presents the order of volatility of data by containing the highest volatile items at the top [30,48,50] :

1. Register, cache
2. Routing table, arp table, process table, kernel statistics, memory
3. Temporary file system
4. Disk or other storage media
5. Remote logging and monitoring data
6. Physical configuration, network topology
7. Archival media (e.g. floppy, backup media, CD-ROM, hard copies)

Before preserving any digital data forensic examiners are advised to understand the volatility of data, in order to apply adequate procedures to the data which needs to be preserved.

## 2.2 Introduction to File Systems

When dealing with the long-term storage of data, such as files on a computer system, file systems are crucial to accomplish this task. File systems are responsible for maintaining various information about files like structural data, e.g. the creation time of a file, the file content [33]. Several file system types on different operating systems are available, for example, the File Allocation Table (FAT), the New Technology File System (NTFS) or the Unix File System (UFS) [33, 73, 79].

### 2.2.1 File System Abstraction Levels

File systems are important to computer systems, because they handle data on a logical level (e.g. files, directories) which is a significant feature for a long-term data storage system. According to [22] such storage systems can be divided top-down into six abstraction levels [22]:

- Physical disk

  Disks represent the physical storage devices for data and can be arranged in many ways. For example, a home computer comprising of a single hard disk or a storage area network (SAN) system with countless disk drives may be possible. Physical disk interfaces like ATA/IDE, SCSI or FireWire can be used for accessing disk content. The geometry of a disk consists of platters with tracks and sectors. Usually each sector has a size of 512 bytes and for each track the sector numbers start with one [31, 33]. In general, disk analysis is not part of a digital examination, because a lot of special training, deep knowledge and expensive equipment is necessary (e.g. electron microscopy) [22].

- Disk volume

  A disk volume, which is described as a collection of numerous sectors, can span one or more physical devices. Volumes can be split up into independent partitions, where each partition comprises of a sequence of consecutive sectors. Depending on the configuration, a single disk can consist of a couple of volumes, where some volumes are partitioned [22, 33].

- File system

  File systems are stated to offer appropriate mechanisms for storing a file's content and its meta data on a media by defining the logical format of files [22, 73]. Furthermore, file systems collect statistical information like the disk quota usage of processes or users. Modern file systems manage a journal which can be advantageous in the case of a digital investigation [33], as digital evidence may be found by inspecting the journal. Additionally, journaling file systems can be used for crash recovery purposes. There are two types of journaling file systems, the ones which keep only meta data logs and the others which store data and meta data to their journals. If a system crash takes place, having a journaling file system will make the recovery process less time-consuming [50].

- Data block

  According to [33], data units represent the smallest, freely allocatable element of a file system to store data. Modern file systems use data units of 4096 bytes (4KB), whereas older file systems use smaller ones, e.g. 512 bytes, 1024 bytes. Data units are referred to as blocks or clusters; the term which is used mostly depends on the surrounding file system. The data content of a file, for example, a portion of a video film or a simple text, is stored to a specific data unit on the file system [22]. Files are a collection of many standardised data units that are connected with each other [33].

- Meta data

  Meta data is commonly known as " ... data about data ... " [ [44], Daniel and Daniel, 2011 ]. File meta data describes various structural properties about files, e.g. the file size, the last modification timestamp, the file owner as well as access control information [22]. All this type of information is stored in logical structures like the Master File Table (MFT), when dealing with NTFS, or in form of inodes on Unix-related systems [33, 73].

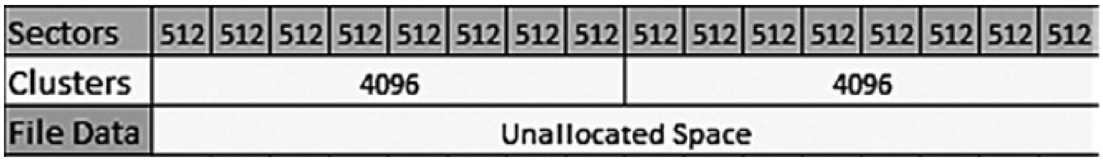
- File

  On the last level there are file names which enable users to find and access various disk content. For example, users can read and write documents or browse directories [22]. Besides their names, files have assigned an identification number which is called inode. The composition of an inode is different on NTFS compared to Unix-based file systems. NTFS uses the Master File Table (MFT) in combination with MFT entries for maintaining its files, whereas Unix-like file systems handle files by using inodes and data blocks. Generally, the term inode is file system independent. Inodes typically contain meta data information about a file together with the pointers to the clusters that are holding the data content [33, 50, 79].

The following section describes the term slack space which is a standard ”feature” of almost all file systems.

### 2.2.2 Slack Space

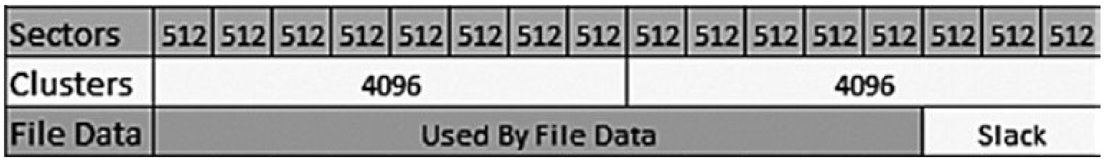
When dealing with file systems, another relevant factor is slack space [79, 89]. As depicted in figure 2.2, modern file systems organise disk space in a multiple of 512-byte sectors (e.g. 1KB, 2KB, 4KB), depending on the volume size [89]. For instance, if disk space is allocated for saving



| Sectors | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Clusters | 4096 | | | | | | | | 4096 | | | | | | | |
| File Data | Unallocated Space | | | | | | | | | | | | | | | |

**Figure 2.2:** File system with 4 KB cluster size contains no data [44]

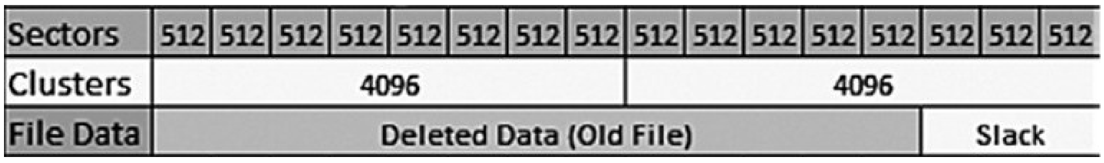
a 6656-byte file, the file system will use two 4096-byte clusters [44]. The remaining 1536 bytes are called file slack space, as shown in figure 2.3. If the file gets deleted, some file systems



| Sectors | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Clusters | 4096 | | | | | | | | 4096 | | | | | | | |
| File Data | Used By File Data | | | | | | | | | | | | | Slack | | |

**Figure 2.3:** File system with 4 KB cluster size contains file data [44]

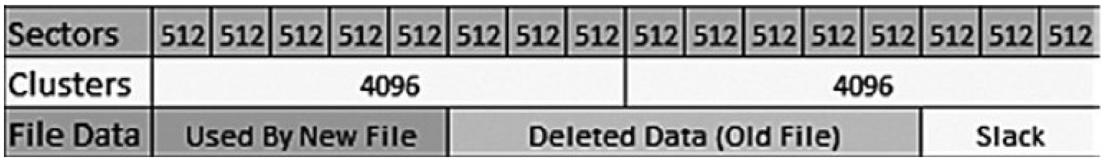
only modify the allocation status of the file-related clusters by changing them from allocated to unallocated [45]. The file content still remains unchanged on the disk which is illustrated in figure 2.4. If a newly created file is to be stored, which is smaller in size than the old file was,



| Sectors | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Clusters | 4096 | | | | | | | | 4096 | | | | | | | |
| File Data | Deleted Data (Old File) | | | | | | | | | | | | | Slack | | |

**Figure 2.4:** File system with 4 KB cluster size after file data has been deleted [44]

the remaining content of the old file will not be overwritten. An example is given in figure 2.5. In contrast to modern operating systems, older versions of Microsoft’s OS, e.g. MS-DOS, did not pad slack space with null bytes [22]. Instead, MS-DOS used variable content from RAM for padding, known as RAM slack [22, 73]. According to [73], file slack plus RAM slack is referred to as drive slack.



| Sectors | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Clusters | 4096 | | | | | | | | 4096 | | | | | | | |
| File Data | Used By New File | | | | | Deleted Data (Old File) | | | | | | | | Slack | | |

**Figure 2.5:** File system with 4 KB cluster size contains new file data and old (deleted) file data [44]

## 2.3 Common Digital Forensic Analysis Techniques

A good knowledge of investigation techniques is essential when performing computer forensics. The following sections introduce common investigation techniques that can be applied to computers in case of a forensic examination. The techniques which have been selected are focusing on Windows platforms mainly, since the practical of this thesis is based upon this operating system.
Digital investigations of such platforms have already been done. For example, Hayes et al. [59,60] discuss general topics concerning Windows forensics, such as biometrics, backup mechanisms, restoration points, touch-screen computing, file encryption, web browsing or file grouping. Additionally, the authors point out the differences between Windows 7, Vista and XP from a forensically stand point of view.

### 2.3.1 Timeline Analysis

In the course of a digital investigation, timelines can be used to reconstruct a map of events that happened on a system. An event can be, for example, the creation of a file. One possible way to generate a timeline can be achieved by using the MAC times of events. The term MAC stands for the time when a file has been modified, accessed or created. Generally the creation time is set when a file is created by saving it on the file system. The modification time is set when the content of a file is altered, whereas the access time is updated on file access.
In the past forensic examinations often focused on timeline analysis of single file systems which nowadays may not be sufficient, due to the fact that an attacker or malicious software leaves multiple traces on several systems, e.g. routers, firewalls. Therefore, constructing overall timelines seems to be a more feasible way. An overall timeline can be created by merging several timelines from different systems to one big timeline, also known as ”super timeline” [ [22], Altheide and Carvey, 2011 ]. By using overall timelines, valuable conclusions can be drawn about events from various systems which may reveal the existence of malicious software or an intrusion taking place [22,34,36,47,87].
General concepts concerning timeline analysis are presented in [22]:

- Relative times

  Despite the fact that system events may be seen as a sequence of dots on a time axis, investigators have to make connections between all these dots and try to understand the relationship between their occurrence. If an examiner focuses only on a few events, timeline analysis may lead to tunnel vision with no reliable outcome. The investigation of relative times can be explained by using three terms: before, after and during. Before and after are used to describe an event that occurred either before or after another event (discrete point) on the timeline. In a similar way the term during can be used to describe that an event occurred between two other events. By considering the concepts before, after or during an investigator is able to reduce the search space for specific events which helps to improve the search accuracy [22].

- Inferred times

  When a full reconstruction of the timeline is not possible, inferred times can be used to determine what happened on a system [22]. The concept of inferred times is usually applied to deleted data. For example, inferring times can be accomplished by comparing the different MAC times of a file. This information can be connected to a file's data clusters, taking into account that some or all data clusters have already been reused [22, 33].

- Embedded times

  Another similar concept to inferred times are embedded times. According to [38], time information can be embedded in many documents like PDF files. If deleted files with embedded timestamps remain unused on the system, digital examiners will be able to extract the embedded times for further analysis by recovering the file content [22].

- Periodicity

  The periodicity of events is another source of information for examiners which indicates the elapsed time between the occurrence of repeated events. This concept enables investigators to recognise backdoor traffic, since malicious software tends to generate network traffic or tries to execute subprocesses in regular intervals. Therefore, fixed-length periods of recurring events or the speed of their recurrence can be used to detect malicious behaviour on a system [22].

In addition to the introduced concepts another term regarding timeline analysis is the "Least Frequency of Occurrence" [ [22], Altheide and Carvey, 2011 ], which means that an examiner should look for rare events, such as two new users that have been added to a rather stable domain of a thousand existing users. This may also be an indicator for a potential intrusion [22].
Many frameworks, such as The Sleuth Kit (TSK) support timeline generation. When creating timelines of NTFS-based Windows systems with TSK, MAC times are displayed by using the MACB pattern, where M represents the modification time, A stands for the last access time, the letter C is assigned to the last modified time of the file meta data and B refers to the time when the file was created.
The initial task when creating a timeline is to locate the target file system. TSK offers a command line utility called mmls that can be used to perform this task, as seen in listing 2.1 [34, 36].

```
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

     Slot   Start        End          Length       Description
00:  Meta   0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----  0000000000   0000002047   0000002048   Unallocated
02:  00:00  0000002048   0000206847   0000204800   NTFS (0x07)
03:  00:01  0000206848   0052426751   0052219904   NTFS (0x07)
04:  -----  0052426752   0052428799   0000002048   Unallocated
```

**Listing 2.1:** Sample output of The Sleuth Kit command line utility mmls describing the structure of a disk image file [36]

As shown in listing 2.1, there are two NTFS partitions contained in the disk image. Within the next step, a NTFS partition is chosen arbitrarily for timeline generation. TSK provides another utility called fls that creates a so called ”body file” [ [34], Carvey, 2009 ], which is a text file with a purpose-built format [34]. A fls call can be performed as follows, as shown in listing 2.2.

```
fls -o 206848 -i raw -f ntfs -r -p -m "C:" image_file > body_file.txt
```

**Listing 2.2:** Sample call of the The Sleuth Kit command line utility fls when creating a body file [34]

The command line options in listing 2.2 are described as follows: -o stands for the offset of the particular partition according to the mmls output, -i specifies the output format, -f refers to the file system type, -r forces the program to traverse folders recursively, -p tells the program to display full paths and -m specifies the mount point (e.g. a drive letter, the root folder) [36]. As mentioned above, the body file has a specific format [34]. The excerpt shown in listing 2.3 shows a schematic view of an arbitrary body file.

```
0|C:/$AttrDef|4-128-4|r/rr-xr-xr-x|48|0|2560|1362002463|1362002463|/
1362002463|1362002463
0|C:/$BadClus|8-128-2|r/rr-xr-xr-x|0|0|0|1362002463|1362002463|/
1362002463|1362002463
0|C:/$BadClus:$Bad|8-128-1|r/rr-xr-xr-x|0|0|26736586752|1362002463|/
1362002463|1362002463|1362002463
0|C:/$Bitmap|6-128-4|r/rr-xr-xr-x|0|0|815936|1362002463|1362002463|/
1362002463|1362002463
0|C:/$Boot|7-128-1|r/rr-xr-xr-x|48|0|8192|1362002463|1362002463|/
1362002463|1362002463
0|C:/$Extend|11-144-4|d/dr-xr-xr-x|0|0|552|1362002463|1362002463|/
1362002463|1362002463
0|C:/$Extend/$ObjId:$O|25-144-6|r/rr-xr-xr-x|0|0|56|1362002465|/
1362002465|1362002465|1362002465
0|C:/$Extend/$Quota:$O|24-144-3|r/rr-xr-xr-x|0|0|88|1362002465|/
```

```
1362002465|1362002465|1362002465
```

**Listing 2.3:** Excerpted content of a body file generated by the The Sleuth Kit command line utility fls [34]

Body files usually can be converted into a human-readable format, for example, by applying a Perl script to the file [34].
When conducting a timeline analysis of NTFS, investigators should be aware that NTFS maintains two different MAC time fields for each file. A detailed analysis of MAC times on NTFS is presented in [25,33,42].

### 2.3.2 Deleted File Recovery

This investigation technique deals with the recovery of deleted files. Usually, for end users the visible data on a file system is more important, than the invisible one. Forensic investigators might be more interested in invisible data, especially data which has been deleted. Most forensic examiners know that deleting data usually does not really delete it, but marks the allocation status of the related data clusters as free. In most cases the file still exists, but is invisible for normal users. Files can be deleted conventionally, for instance, by using the DEL or ERASE command on Windows. Moving files may also delete them, for example, if a file is moved from one hard disk to another [34,89].
Salvaging deleted data can be accomplished manually or by adopting automated recovery tools, but care should be taken, due to the false positive rate of automatically recovered data. It is suggested that forensic examiners verify the correctness and consistency of the generated tool output and match the results with other file system details [38]. According to [22], deleted data can be classified more precisely by using the following terms [22]:

- Deleted data

  This type of deleted data is represented by an unlinked file (or directory) that is not visible on the user level. Despite that, the file has got an intact header, which stores the file name and additional meta data. The file content still remains on the file system and is connected to the file header, except that the file space is flagged as free for recycling [22].

- Unallocated data

  A file is called unallocated when the file meta data structure (e.g. the MFT entry) was unallocated or reused, but the file content (e.g. the clusters) is still remaining in data clusters on the disk. File carving is a method which can be used to export idle data clusters from the file system [22,37].

- Orphaned data

  Deleted data is called orphaned when the link between the data and meta data structure of a file (or a directory) is not working anymore. File recovery may not be possible and result in inaccurate findings [22].

- Overwritten data

  ”Files that are overwritten are generally considered to be unrecoverable.” [ [87], Sammons, 2012 ] From a forensically standpoint, the recovery of overwritten files is a challenging task. A full recovery is not possible due to the fact that at least one or more clusters of a file have been reallocated to other files. Overwritten files can only be recovered partially depending on the extent of overwriting [22].

Users who are not fully aware of how their computer works may think that pulling data into the recycle bin will delete it. The recycle bin may be the first place to look at when collecting digital evidence. Users can bypass the recycle bin in Windows by setting the NukeOnDelete key in the Windows Registry or by hitting Shift and Delete. In contrast to prior versions of Windows, Windows 7 has reorganised the recycle bin structure. When a file is deleted through the Windows Explorer shell, it will be deferred to a subdirectory of $Recycle.Bin. This sub folder is named after the current user’s SID. Files sent to $Recycle.Bin keep its original file extension. The file names start with $R followed by six various characters [34, 36, 87].
After inspecting the recycle bin thoroughly, in order to detect digital evidence, an investigator may consider the next step to be the recovery of MFT entries. Data can be recovered unless the MFT entry has not been reused, because the file system does not necessarily delete MFT entries once they have been created. For example, when a large amount of files is being created and deleted, NTFS will keep the corresponding MFT entries for an indefinite period of time. In fact, NTFS will attach new entries to the MFT or reuse old ones for storing new data. When a file is deleted, its MFT entry will be set to unallocated as well as its data clusters. The recovery of unallocated files is possible by parsing the entire MFT. Unallocated MFT entries usually have a reference to their parent directory. Therefore, the whole file path can be reconstructed. Unless the MFT entry of the parent directory has been reused or unless the file content has been overwritten, a full recovery of the corresponding file is likely to be successful [22, 33, 37, 89].
For an investigator it may be also interesting, when a file or directory was deleted or by whom it was deleted. As there is no update of the timestamps of deleted files when they are removed, the deletion time only can be approximated from the time the file was last used or modified. It is also rather difficult to say what user deleted a specific file. Therefore, consistency checks need to be done, for example, by including the verification of the file permissions or by examining the most recently used files [38].
Beside the aforementioned file recovery strategies, there exist other ways to reconstruct deleted data as well. An approach is proposed in [72], where an algorithm was used to rebuild directory trees for deleted files.
Other data recovery approaches can be accomplished by using The Sleuth Kit. This framework offers several command line utilities to recover deleted data, such as ils, icat, ifind or ffind. For instance, the ils utility lists unallocated files by default, whereas icat is used to recover the file content of inodes. The ffind program is used to map the inode number to the file or directory name that references it - ifind maps data clusters to their corresponding inode [50].

### 2.3.3 Windows Registry Forensics

Unlike Unix-based operating systems, a core component of modern Windows OSes is the registry which is a hierarchical database that maintains various settings regarding OS users and (un)installed applications, as well as plenty of configuration data about the operating system itself. Configuration data can be time-based information, for example, the date when a user last logged in a wireless network. There exists also other configuration data, for instance, in Windows Vista there is a setting that tells the operating system not to update a file's last access time, which is enabled by default. From a forensic standpoint the Windows Registry holds lots of valuable information about a considerable amount of events which happened during runtime. As a result of the large amount of data contained in the registry and due to the fact that Windows is of closed source, forensic analysis can be challenging, since there is hardly anybody who has highly detailed knowledge of every registry entry [34–36].
Another obstacle concerning a digital investigation of the Windows Registry is the fragmentation of the registry entries. Usually registry fragmentation results from the software which was installed or removed. Investigators will be confronted with invalid and inconsistent registry entries [62]. Therefore, general approaches should be considered when analysing the Windows Registry. For instance, the principle of the "Least Frequency of Occurrence" [ [36], Carvey, 2012 ] can be applied to detect malicious system behaviour. Even investigators should have an eye on most recently used registry entries or entries that reveal information about the URLs typed by a user into a web browser [34–36, 38].
The Windows Registry is organised in files, known as hives, as well as keys and subkeys which are explained below. Some of this hive files are stored on the system permanently, while others are volatile having their content accessible only when the system is up. Investigators should keep this in mind and extract the necessary data from the volatile hives, especially when performing a post mortem analysis [34–36].
In accordance with [34–36, 40], the core hives in later Windows versions like Windows 7 comprise of the files SECURITY, SAM, software and system. Additionally, beginning with Windows 2000 another hive file, called Default, was released, which is a subkey of HKEY_USERS. As compared with modern Windows OSes (e.g. Vista, 7), prior versions of Windows (e.g. 95, 98) stored some of the hives differently. For example, in Windows 7 the hives which belong to the registry key HKEY_LOCAL_MACHINE (SAM, SECURITY, system and software) can be found in the directory Windows\system32\config, whereas in Windows 98 the hive files were located in the Windows root folder. These hive files used names dissimilar from Windows 7, for example, in Windows 98 there was a hive file named user.dat [35, 37]. In order to analyse the Windows Registry, forensic examiners can access the hive files by using specific keys and subkeys [34]:

- HKEY_LOCAL_MACHINE

  The keys contained in this hive are composed out of the core hive files. This hive maintains the central configuration of the system, such as hardware drivers or diverse software settings [34–36].

- HKEY_USERS

  The hive associated to this name manages the actively loaded user profiles for the computer system. Typically, the user profiles are located within the NTUSER.DAT hive file. With Windows 7, some entries have been moved from the NTUSER.DAT hive to the USRCLASS.dat hive. Both hives are merged when a user logs in, in order to be presented as unified file [34–36].

- HKEY_CURRENT_USER

  The keys contained in this hive are volatile and pertain to the currently logged-on user. For example, there is a key counting the number of running programs on the desktop [34–36].

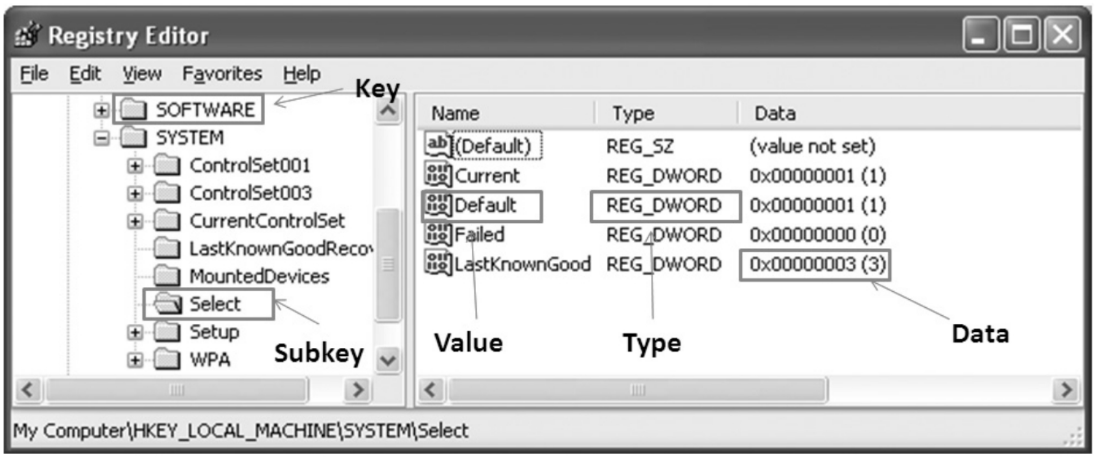
- HKEY_CURRENT_CONFIG

  This hive is volatile as well and represents the hardware profile which is used on system start-up [34].

- HKEY_CLASSES_ROOT

  This hive is volatile in nature and maintains the file types on a Windows system. Additionally, it determines the application to be used to open specific file types. HKEY_CLASSES_ROOT is initialised based upon the hives HKEY_LOCAL_MACHINE and HKEY_CURRENT_USER [34–36].

As illustrated in figure 2.6, the registry is structured hierarchically and consists of keys and subkeys containing different values. Generally, every value has the attributes type and data. The type attribute can have one out of multiple values. Figure 2.6 shows two types named REG_SZ and REG_DWORD where the former represents a fixed length text string (e.g. Unicode, ASCII) and the latter indicates a 4 byte integer number. Having cognisance of the different type values will enable investigators to perform an effective search. In fact, the value range of the data attribute depends on the type attribute which was chosen when modifying a specific registry key [34].



| Name | Type | Data |
|---|---|---|
| (Default) | REG_SZ | (value not set) |
| Current | REG_DWORD | 0x00000001 (1) |
| Default | REG_DWORD | 0x00000001 (1) |
| Failed | REG_DWORD | 0x00000000 (0) |
| LastKnownGood | REG_DWORD | 0x00000003 (3) |

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\Select

**Figure 2.6:** Sample view of the Windows Registry Editor showing keys, subkeys, values and types [35]

The Windows Registry can be analysed in various ways, either investigators can connect to a computer system doing a live scan or the registry can be examined offline. For analysis purposes the usage of tools (e.g. Regshot) is recommended due to the complexity of the registry data [34–36]. Investigators may also conduct a manual examination, since there are registry keys which are commonly known to bare valuable information. For example, the HKEY_LOCAL_MACHINE\Software key contains subkeys which display the patch level, the current version, the installation date or the Microsoft product ID of the operating system. Other subkeys included in HKEY_LOCAL_MACHINE\System\ControlSet\Control hold information about the computer name, the time zone, the log on/off time or the time of the last shutdown [40, 47].
Besides, in [97] additional research has been done in the field of Windows Registry forensics by pointing out particular registry keys that can be used especially in case of an intrusion. The authors have concluded that the valuable registry keys to be analysed may vary based on the case [97].

### 2.3.4 File Carving

Recovering deleted data does not always reveal valuable information. There are some reasons why investigators may struggle to find digital evidence. For example, traditional data recovery will not be possible, if the file meta data of deleted files, which has been partially or fully overwritten with other data, no longer exists. Therefore, an advanced data recovery approach, called file carving, can be used to extract and reconstruct deleted data from unallocated space [38, 44].
Usually file carving is based upon the analysis of file headers and footers [37, 38, 87]. This type of analysis is also referred to as file structure-based carving [77]. Both file header and footer are also referred to as file signature. In general, each file has a header as well as a corresponding

footer. In between the header and footer the file data is stored. Figure 2.7 shows a list of common file signatures. Once a file's header or footer is detected, it can be carved out partially or fully

| File Type | Header | Footer |
|---|---|---|
| JPEG | Usually `FF D8 FF E0` or `FF D8 FF E1` and sometimes `FF D8 FF E3` | `FF D9` |
| GIF | `47 49 46 38 37 61` or `47 49 46 38 39 61` | `00 3B` |
| Microsoft Office | `D0 CF 11 E0 A1 B1 1A E1` | N/A |

**Figure 2.7:** Sample file signatures for JPEG, GIF and Microsoft Office files [37]

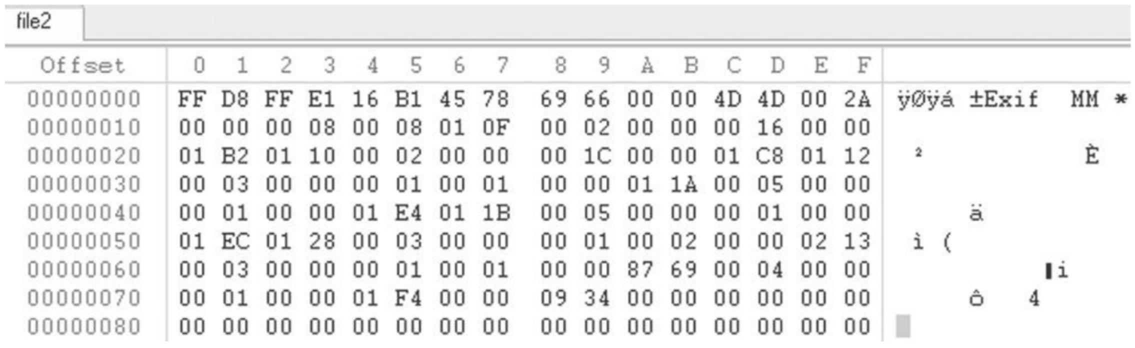
depending on the integrity of the file, in other words, the effectiveness of file carving relies on the assumption of unallocated files having arranged their data in contiguous clusters [37,38,87]. Despite the fact that modern operating systems try to store data on the hard drive contiguously, there are some reasons why the file system is forced to write files which comprise of two or more fragments [51]:

- Data on a file system will be written in fragments, if the hard drive has been used for a long time. There may be no consecutive clusters to store the data, because lots of data has been added and removed.

- When appending data to a file, the additional data will be stored to a different location on the hard disk due to the lack of unallocated space behind the original file. This behaviour was observed on most file systems, but sometimes files will be completely relocated.

- Depending on the file system type, file fragmentation is a standard feature, for example, UFS will store big files in fragments to the hard drive.

File carving can be performed manually, e.g. via copy and paste, or automatically by using tools like foremost or scalpel. Furthermore, investigators will find digital evidence by inspecting a file with a hex editor manually [37]. As depicted in figure 2.8, the characteristic string Exif denotes that the file is stored in the Exchangeable Image File Format - commonly used on digital cameras. Prior to Exif the file header displays the string ÿØÿá, which is related to the JPEG file header FF D8 FF, as shown in figure 2.7. This type of validation method is also called ”Magic Number Matching” [ [23], Aronson and van den Bos, 2011 ]. Other validation approaches may consider using some of the following aspects to detect unallocated data, such as file dependencies (e.g. size, internal structure), cyclic redundancy check sums, different file encodings or compression formats [23].
Typically, the process of salvaging files by means of tools generates a significant amount of fragmented or broken files. Due to the lack of file meta data, in some cases, recovery tools can only estimate the original size of the files to be recovered. To overcome this problem, partial file carving can be accomplished instead of salvaging a full disk image. Partial file carving takes advantage of restoring and merging the different data parts of a file into a single file, which would



```
file2
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000  FF D8 FF E1 16 B1 45 78  69 66 00 00 4D 4D 00 2A  ÿØÿá ±Exif  MM *
00000010  00 00 00 08 00 08 01 0F  00 02 00 00 00 16 00 00
00000020  01 B2 01 10 00 02 00 00  00 1C 00 00 01 C8 01 12   ²          È
00000030  00 03 00 00 00 01 00 01  00 00 01 1A 00 05 00 00
00000040  00 01 00 00 01 E4 01 1B  00 05 00 00 00 01 00 00       ä
00000050  01 EC 01 28 00 03 00 00  00 01 00 02 00 00 02 13   ì (
00000060  00 03 00 00 00 01 00 01  00 00 87 69 00 04 00 00            ▮i
00000070  00 01 00 00 01 F4 00 00  09 34 00 00 00 00 00 00       ô   4
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

**Figure 2.8:** Sample view of a hex editor displaying the file signature of a JPEG file [37]

allow for more control of the recovery process than carving the full image [37, 38, 87].
Similar to file structure-based carving fast object validation can be applied to carve bifragmented files. Fast object validation deals with recovering files consisting of two fragments, where each fragment is decoded and validated by using the bifragment gap carving algorithm. In order to apply this method properly, working file headers and footers are necessary [51].
Several different concepts and techniques regarding file carving have been presented in [77] to circumvent the weaknesses of carving methods base upon the file structure. Carving techniques have been developed which have adopted graph theory for problem solving. For example, file carving has been formulated as Hamilton path problem. The k-vertex disjoint path problem has resulted from the refinement of the Hamilton path problem, as the prior one has not taken file fragmentation statistics into account. Additionally, graph-based algorithms, such as parallel unique path and shortest path first, have been proposed [77].
Towards graph-based techniques, another methodology has been proposed in [83] which uses file fragment classification in order to find files that match specific patterns of data formats like JPEG, MP3 or ZLIB. As the authors have concluded, their approach is based on easy discoverable patterns. The authors have emphasised to focus on the establishment of specialised strategies, in order to target only on certain data.
Further research has yielded in various scientific papers that propose methods for carving graphics files as well as compressed data. As shown in [71], the accuracy of image and thumbnail file carving has been improved by using a new pattern matching approach. Another study [65] on image files focuses on the calculation of similar pixels which are contained within different image file fragments. It is also possible to recover JPEG-encoded images by using the corresponding thumbnail files from the thumbnails database, which usually is created when the image is viewed [55]. Other studies developed file carving methods for compressed data, especially for RAR archives [96] and NTFS compressed data [98].
File carving has been applied to the Windows Registry as well by parsing the internal structure of the hive files, which are binary by nature and organised on the disk in so called HBIN blocks, in order to reassemble the registry keys. Due to the fragmentation of the registry hive files, only not fragmented hive files have been recovered [95].

### 2.3.5 Network-based Forensics

Towards forensic investigation techniques like deleted file recovery or file carving, the area of network-based forensics is more sophisticated, because investigations are not limited to a local computer system, but will include many computers from several networks (e.g. company and/or home networks), which can be connected to each other differently, e.g. over wire or wireless. Network-based forensic methods typically focus on wired networks, but due to the popularity of wireless networks, cyber criminals may abuse them as well to perform illegal activities, such as launching attacks against other network users [49, 87]. By applying digital forensics to wireless networks, these issues can be overcome [68].
Another hurdle concerning network-based forensics is that capturing all traffic may not be possible due to the complexity of the network structure or the large amount of data which flows through it [49, 87]. To improve network-based examinations Sibiya et al. [90] present guidelines which cover best practises to establish a forensically sound investigation process.
When investigating the network, examiners should be familiar with the different network types that exist, at least the Local Area Network (LAN) or the Wide Area Network (WAN).
An important topic regarding network types is packet routing. Packet routing deals with the transmission of small data chunks, called packets, which are exchanged between a sender and a receiver. These packets may cover digital evidence and capturing them may possibly reveal valuable information to an investigator. Forensic examiners may be able to extract sensitive data collected from network traffic like user names or passwords [49, 87]. Different approaches for packet analysis are presented in [90], such as protocol analysis, packet attribute extraction or charting the network flow. Moreover, by monitoring the network regularly, an examiner may be able to uncover a network intrusion taking place [49, 87]. As presented in [88], forensic investigators were able to detect a spam network by monitoring Domain Name System (DNS) traffic.
Usually computer-based networks are build on the well known OSI reference model by which data is divided into small packets before transmission. There exist a variety of network protocols. Among them there is the well known Transmission Control Protocol (TCP) which is build upon the Internet Protocol (IP). Since the Internet Protocol is part of the OSI reference model, investigators are recommended to get acquainted with the different OSI layers before performing packet analysis [66]. To get a better understanding of the complexity of IP packets, figure 2.9 depicts an IP packet with its various data fields, such as source and destination IP address, as well as the total length of the IP packet. Typically, capturing and analysing network traffic, such as IP packets, can be accomplished by using tools like tcpdump or its more popular extension for Windows systems, called Wireshark, which additionally provides a graphical user interface [66]. Wireshark is a capable of collecting live network traffic. It can read traffic from different interfaces like Ethernet, IEEE 802.11 (wireless) or FDDI. Additionally, it provides investigators with useful features to reconstruct the data that is located on the application layer of the OSI model [49, 66].
Another packet acquisition and analysis software similar to Wireshark is presented in [56], where a program has been implemented which automatically captures all packets from the network and extracts the transmitted data consecutively.
Beside the examination of network traffic, digital evidence may be contained within log files as well, e.g. web server logs. Investigators may be able to retrieve additional information from

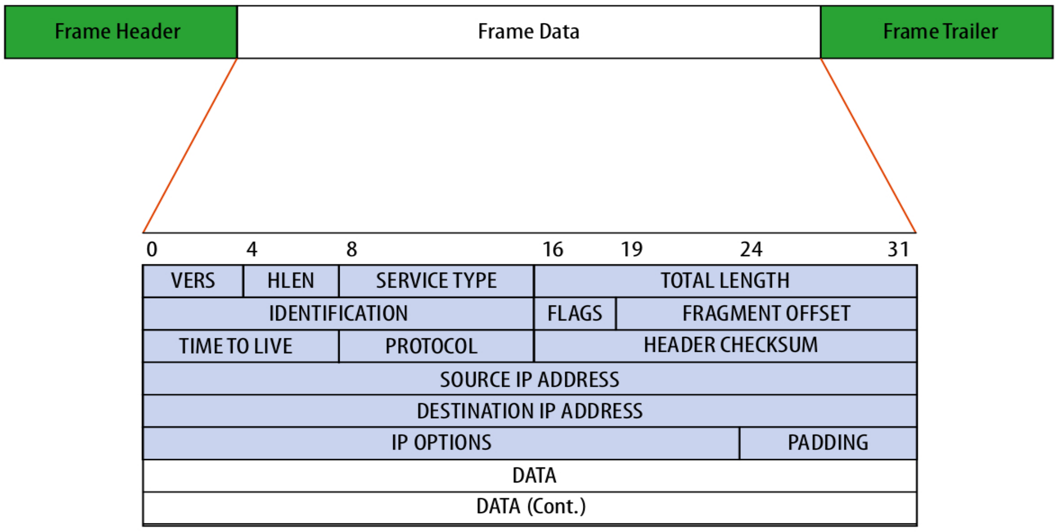


**Figure 2.9:** Sample view of an IP packet comprising of various attributes [66]

log file inspection like legal and illegal user actions which have been performed on a computer system [49, 87].

### 2.3.6 Physical Memory Forensics

In accordance with [31,37], volatile data can be contained in the physical memory of a computer, for example, in the RAM. Many operating systems extend the physical memory by a logical (virtual) memory which is stored on the hard disk, for example, in form of page or swap files. In general, physical memory data disappears when the system is shut down, whereas logical memory data perhaps will remain on the hard disk depending on the system configuration [31]. Therefore, it is advisable to capture the data most likely to be altered, such as the data contained within the physical memory [94].
In contrast with physical computing environments, the memory handling of virtual machines (VMs) is different, because the content of the RAM is stored on the hard disk when the VM goes in suspended mode. This allows investigators to collect the physical memory from hard drives [38].
Analysing volatile data either from physical or logical memory often discloses important information about what happened on a live system. A forensic investigator may encounter the following information, for example [31, 37, 61, 94]:

- Current network information (e.g. TCP/UDP ports or IP address information)
- Processes currently executed (i.e. loaded libraries referring to their parent process)
- Tasks currently scheduled (i.e. task flagged as completed)
- Clipboard data (i.e. user input buffered for further usage)

- Open file handles (i.e. files opened by processes for reading/writing)

The items provided in the list above is not exhaustive. One can imagine that a forensic examiner can find other volatile data in memory as well, such as malware (e.g. rootkits). Typically, malicious software can be detected when it is loaded into physical memory for execution or written to swap space [31, 37].
Usually before a physical memory analysis can be started, the memory has to be dumped to a file. Physical memory images of a live system are created in a similar way to which a hard disk of a dead system is acquired. There are many tools that can be adopted for physical memory acquisition like dd, Winen or Fastdump. Besides, the destructiveness of memory acquisition tools should be taken into account, i.e. a collection tool running on Windows XP may affect the physical memory regarding the memory needed by the acquisition tool itself, as well as the Dynamic Link Libraries (DDLs) that are needed for executing this tool. In order to choose an appropriate memory acquisition tool which has only little effect on the memory, several quality criteria can be defined, such as physical memory consumption (e.g. page file count), file system impact (e.g. shared DLLs count) and virtual memory usage (e.g. virtual byte count) [31, 34, 61, 94].
After dumping the memory to a file there are various ways for content analysis like doing a keyword search, running the utility software strings on a dump file or browsing the dump file with a hex editor [24].
In contrast to a keyword search that is conducted manually, forensic frameworks, such as Volatility or Memoryze, can be used to improve the outcome of a physical memory investigation [34]. Volatility is able to determine the operating system, for example, by supplying the ident command line option, as illustrated in listing 2.4.

```
F:\MemoryForensics>python volatility ident -f F:\image\xp-suspicious.dd
         Image Name      : F:\image\xp-suspicios.dd
         VM Type         : Service Pack 1
         Image Type      : nopae
         DTB             : 0x38000
         Datetime        : Thu Jul 07 16:34:23 2011
```

**Listing 2.4:** Sample output of the command line tool Volatility identifying the operating system on the basis of a physical memory dump [34]

Other command line options like plist or connections can be specified as well. The plist option allows investigators, for instance, to retrieve the list of running processes from a memory image, whereas the connection option lists all active network connections [34]. An excerpt of Volatility's plist output is shown in listing 2.5.

```
F:\MemoryForensics>python volatility pslist -f F:\image\xp-suspicious.dd
Name             Pid     PPid    Thds    Hnds   Time
Fast.exe         301      201       3      21   Jul 27 17:46:30 2011
winlogon.exe     302      202      15     596   Jul 27 17:46:31 2011
TaskSwitch.exe   303      203       5     583   Jul 27 17:46:32 2011
Directcd.exe     304      206      45      23   Jul 27 17:46:33 2011
```

```
services.exe     305      209       7      293   Jul 27 17:46:34 2011
ati2evxx.exe     306      223      20      259   Jul 27 17:46:35 2011
Crypserv.exe     307      244      32      694   Jul 27 17:46:36 2011
EM_EXEC.exe      308      225      83      192   Jul 27 17:46:45 2011
atiptaxx.exe     309      230       1      204   Jul 27 17:46:46 2011
Smc.exe          310      231      34      506   Jul 27 17:46:47 2011
mqsvc.exe        311      261      22      539   Jul 27 17:46:48 2011
```

**Listing 2.5:** Sample output of the command line tool Volatility listing the running processes of a Windows operating system based on a physical memory dump [34]

In contrast to frameworks like Memoryze or Volatitily, the authors of [99, 100] have proposed another method based upon the "Kernel Processor Control Region" [ [100], Zang et al., 2010 ] to extract, for example, the list of running processes or user login information from the memory of a Windows 7 system.
Other studies are focusing on the physical memory of Windows as well and have proposed an approach which deals with keyword searching. The search patterns comprise of the user input that is supplied to the application during runtime. Memory images have been evaluated of the following Windows applications in the course of a digital investigation: Excel, Outlook, PowerPoint and Internet Explorer. The results have shown that the applications have stored many string and numeric values to the physical memory. For example, salvaging numeric values of Excel has been difficult due to the numeric format that has been used in memory [75, 76].

## 2.4 Digital Anti-Forensic Techniques

As forensic techniques evolved within the last decade, anti-forensic techniques did as well. In general, anti-forensic techniques can be used to confuse forensic examiners with the aim to hamper the digital investigation process. For example, criminals or terrorists used anti-forensic methods to wipe out their digital traces. Even the corporate management took advantage of anti-forensic tools in order to hide electronic records, such as suspicious e-mails [87].
The term anti-forensics can be defined in various ways. According to Barbara, anti-forensics is defined as " ... an approach to manipulate, erase, or obfuscate digital data or to make its examination difficult, time consuming, or virtually impossible." [ [87], Barbara (as cited in Sammons), 2012 ]. Since anti-forensics provide a large variety of techniques, only commonly-used ones will be described within the next section to get a good basic understanding.
In accordance with [43], anti-forensics techniques can be divided into the categories data wiping, data corruption and data hiding, whereas the latter one is most commonly used. According to [28,34,37,39,45,46,67,70,73,79,87,91–93], data can be hidden by making use of the following techniques: data encryption, hidden directories and files, deleted directories and files, renamed directories and files, steganography, alternate data streams, etc. Data hiding and obfuscation techniques targeting the physical structure of a hard disk, such as the "Host Protected Area" [ [27], Berghel, 2007 ] (HPA) and the "Device Control Overlay" [ [27], Berghel, 2007 ] (DCO), are discussed in [27, 37] as well.
In contrast to data hiding, data wiping as well as data corruption deal with the manipulation and

destruction of data. Although the underlying intention of data destruction seems to be noble, since this technique can be used to preserve user privacy, misuse is possible, in order to defeat data recovery techniques [29, 87]. As data destruction is not limited to a local system, forensic examiners should be aware of remote wipes a forger could perform [80]. The quality of data wiping mostly is based upon the wipe algorithm and the number of deletion runs that were applied to the data. Consequently, there is no guarantee that a wiping software will destruct all data that needs to be deleted.

Defragmentation can be also used to corrupt or overwrite data. This process involves tying up the clusters related to a file and moving them to another disk location, in the hope of destroying data with evidentiary value. The effectiveness of this approach is considered to be fair, since not all valuable data may be deleted [29, 87].

In [78] the authors present a method of detecting and erasing sensitive data stored on hard disks. Implemented as a Windows service, the proposed solution determines the sensitivity of data by means of user-defined classification rules, file system monitoring (e.g. copy, modify or delete files) and disk scanning (e.g. initial, maintenance). The results have shown that sensitive data remained undetected on the hard disk.

Another opportunity to wipe sensitive data permanently is provided by means of encryption with secure key erase. A drawback regarding this technique is that the data will not stay encrypted forever. Data recovery will be possible, if the encryption method initially used for encryption was broken [46]. Therefore, forgers may adopt more powerful ways of data destruction which include procedures like demagnetising, etching or shredding media [89].

## 2.5 Digital Forensic Analysis of Modern Web Browsers

The major topic introduced within this section is the digital investigation of the private mode of modern web browsers. Private web browsing deals with data hiding, data corruption as well as data wiping, such that it can be considered as a part of digital anti-forensics.

### 2.5.1 Private Browsing Mode

Beside the standard features offered by web browser vendors, a feature implemented in newer browser versions is called private browsing mode. The purpose of this feature is to protect users from being tracked on local computer systems, e.g. during a forensic investigation. For example, in the newer versions of Firefox private browsing will not save user-related data, such as the visited web pages. Moreover, individualised data contained in the browser cache as well as passwords, cookies, search bar entries and form field entries will no longer be stored [87].

### 2.5.2 Related Work

As aforementioned, a major issue concerning web browsers is user privacy. In [32] a study is presented about the safety of web browsers. The results have shown that popular web browsers like Internet Explorer and Firefox contain many vulnerabilities that might have affected the private browsing mode as well. Due to the fast development of modern web browsers newer software versions will contain vulnerabilities which will have an adverse effect on user privacy.

As a consequence, digital investigation of the private browsing mode has been done in order to ascertain whether user-related data has been leaked to the local computer or not.
In [58], an investigation approach is presented which has been applied to the Internet Explorer. The study has focused on the examination of user-related data which has been located in the unallocated disk space, such as cookies, browsing history and cache. Although the approach has been implemented thoroughly, a disadvantage is that this investigation technique is not scalable, since it only targets one web browser. Another drawback of this method is that it uses a keyword search which is limited on a couple of pre-defined string values for parsing the unallocated disk space. In [20], a similar approach has been applied to Facebook's instant messaging service in order to detect digital remnants left on the local hard drive with respect to user-related chat messages.
In contrast, the automated monitoring of web URLs for forensic analysis is presented in [19]. A HTTP-sniffer has been proposed for logging individualised browsing data by storing it to a remote server. An advantage of this methodology is that the digital examination of the web contents accessed by a user is possible even if the local hard disk has been wiped. The main issue concerning this solution is that HTTP traffic is logged only, regardless of where and what user-related data has been written to the local hard drive.
In [63], a tool is presented that is capable of reconstructing and visualising a user's browsing session by taking advantage of the Internet Explorer API. Although the visualisation of the browsing data may be useful for digital investigators in order to rebuild a user's browsing session, the approach only deals with allocated data, but neglects the unallocated disk space. Since this study has focused on the Internet Explorer, user-related session data of other web browsers cannot be recovered by using this tool.
In contrast to the aforementioned studies which have not explicitly discussed private web browsing, the following studies survey the investigation of the private browsing mode. The digital investigation of portable web browsers like Chrome is presented in [69]. The forensic artefacts, which remain locally after browsing in public and private mode, have been evaluated. A keyword search similar to the proposed method from [58] has been applied.
In [86] and [18] the private browsing mode of popular web browsers (Internet Explorer, Firefox, Chrome, Safari) has been examined. Different approaches have been used in [86], such as a keyword search as well as the forensic analysis of the physical memory and the hard drive. Furthermore, [18] evaluates different browser add-ons of the aforementioned web browsers and conducts a source code analysis for Firefox as well as automated unit testing. It has been found that web browser add-ons can threaten user privacy. Despite this fact, both studies [18, 86] have not revealed any information about whether they have been able to monitor all (from the start until the end) browser actions which have been performed during a private browsing session or not.
In [74], further research has been done on private and portable web browsers. In opposition to [18, 86], this approach uses a utility, called DaemonFS, which is capable of monitoring the actions a software application carries out on the file system, e.g. a web browser. For further analysis FTK Imager has been utilised. As the results have shown, individualised data has been located in the physical memory, in the unallocated space and in the slack space. In [41], a detailed forensic analysis regarding the private browsing mode of the Internet Explorer 10 has been

done. The study has shown that the Internet Explorer has stored lots of user-related browsing data locally, over different series of application live cycles. Another thing worth mentioning is that the digital investigation has taken advantage of virtual machines, in order to run different tests on the Internet Explorer.
Although much excellent research has been done, further research is still needed.

CHAPTER 3

# Conceptual Design

This chapter presents a new approach for the digital investigation of the private browsing mode of modern web browsers. The theoretical concepts of virtualisation and process monitoring are described, as they represent the two fundamental parts which the proposed investigation approach is based on.
Beside an introduction to the field of virtualisation and process monitoring, particular features of both Oracle's virtual machine VirtualBox [17] and Microsoft's process monitoring tool Procmon [11] are explained, which are required by the new investigation approach.
VirtualBox has been chosen as virtual environment for testing, because it is freely available and the provided functionality is sufficient for the investigation approach. Although Windows is not free of charge, it is used as test operating system due to its large user base and because privacy issues will affect the majority of end-users.
Since Windows is the target testing platform, Microsoft's Procmon is the tool of choice for performing process monitoring. This tool is capable of observing process features at a very low-level, e.g. the process memory consumption. Additionally, it provides features for tracing file system events induced by individual processes, e.g. web browsers. When thinking of common digital investigation techniques, process monitoring seems to be more advantageous towards the timeline creation, since a process monitoring log file contains only particular process events, whereas a traditional timeline covers even more system events, such as the operating system noise. Due to the accuracy of process monitor log files, deleted file recovery will be more effective as well. Therefore, process monitoring is chosen as suitable concept for the new investigation approach.

## 3.1 Virtualisation in the Field of Digital Forensics

As virtualisation has grown steadily within the last decade due to benefits like reduced hardware and power costs, the digital investigation of virtual environments has become more popular. Digital examiners have been required to inspect virtual machines (VMs) in order to find digital evidence, because virtual environments are rather likely to be detected and compromised, for example, by an attacker or a malicious software. Contrary to usual investigation scenarios regarding virtual environments, for instance, in the context of a legal dispute, virtual machines can be used for different purposes as well, e.g. software testing. Virtual machines are suitable in this case, since they offer a ”clean” environment that can be used iteratively [26].

### 3.1.1 Virtual Machine Snapshots

Virtual environments typically provide a feature which is referred to as ”snapshot” [ [26], Barrett and Kipper, 2010 ]. A VM snapshot represents a particular virtual machine state that has been captivated at a specific date and stored to an arbitrary media, e.g. a hard drive. In other words, snapshots can be used for preserving (”freezing”) the state of virtual machines which implicates the current state, the settings and data contained on the file system. Reversely, this means that snapshots can be restored arbitrarily at any time, for example, in the case of an error. [26]
Managing snapshots can be accomplished by the usage of ”Universal Unique Identifiers” [ [17], Oracle, 2013 ] - or in its abbreviated form UUIDs. In general, a UUID is a randomly generated alpha numerical string which is commonly used to identify VM snapshots [17, 26].
When investigating virtual environments, acquiring a VM’s file system is a crucial task forensic investigators are concerned with in order to perform further analysis. Although imaging of virtual machines is accomplished similar to ”real” (hardware-based) machines, in the case of a virtual machine only the virtual machine itself needs to be preserved instead of the entire hard disk of a physical computing environment [26].
In order to perform this task, virtual machines offer an imaging functionality which is called ”cloning” [ [26], Barrett and Kipper, 2010 ]. For example, Oracle’s VirtualBox provides the command VBoxManage clonehd for disk imaging (cloning). A sample call of this VBoxManage feature is presented in listing 3.1.

```
VBoxManage clonehd snapshot_UUID.vdi output_file.dd --format RAW
```

**Listing 3.1:** Sample call of the VirtualBox command line utility VBoxManage to convert a virtual machine snapshot into a disk image file [17]

As can be seen in listing 3.1, the instruction clonehd requires three parameters, in order to make VBoxManage work properly. Firstly, the UUID of the snapshot has to be specified plus the suffix .vdi, since VirtualBox maintains its snapshots by means of VDI files [17]. Secondly, the output file name is required which can be chosen randomly. Thirdly, the image format has to be specified, for example, raw indicates that bit copying is going to be performed.

### 3.1.2 Virtual Machine Shared Folders

Virtual machines offer a feature called shared folders, which can be used to transfer data between the guest file system (virtual environment) and the host file system without having a network connection [26]. For example, VirtualBox implements this feature by using purpose-built file system drivers. When creating a virtual share usually a directory is made on the host file system and linked to the guest file system afterwards. Listing 3.2 depicts a sample call, where a shared folder test_share is created on file path /home/shared [17].

```
    VBoxManage sharedfolder add "test_vm" --name "test_share"
                                          --hostpath "/home/shared"
```

**Listing 3.2:** Sample call of the VirtualBox command line utility VBoxManage to create a shared folder [17]

VirtualBox offers the possibility to mount a shared folder manually or automatically after machine start-up. Additionally, shared folders can be mounted in read-only or read/write mode [17].

### 3.1.3 Virtual Machine Screenshots

Typically virtual machines provide a screenshot function which can be used to take screenshots of a guest environment and store them, for example, to a directory on the host system. A VirtualBox screenshot is typically created by typing the VBoxManage command controlvm, as presented in listing 3.3.

```
    VBoxManage controlvm "vm_name" screenshotpng "foto.png"
```

**Listing 3.3:** Sample call of the VirtualBox command line utility VBoxManage to create a screenshot [17]

Virtual machine screenshots can be used for diagnostic purposes, for example, by taking a picture of the virtual environment's GUI after a program has crashed [17].

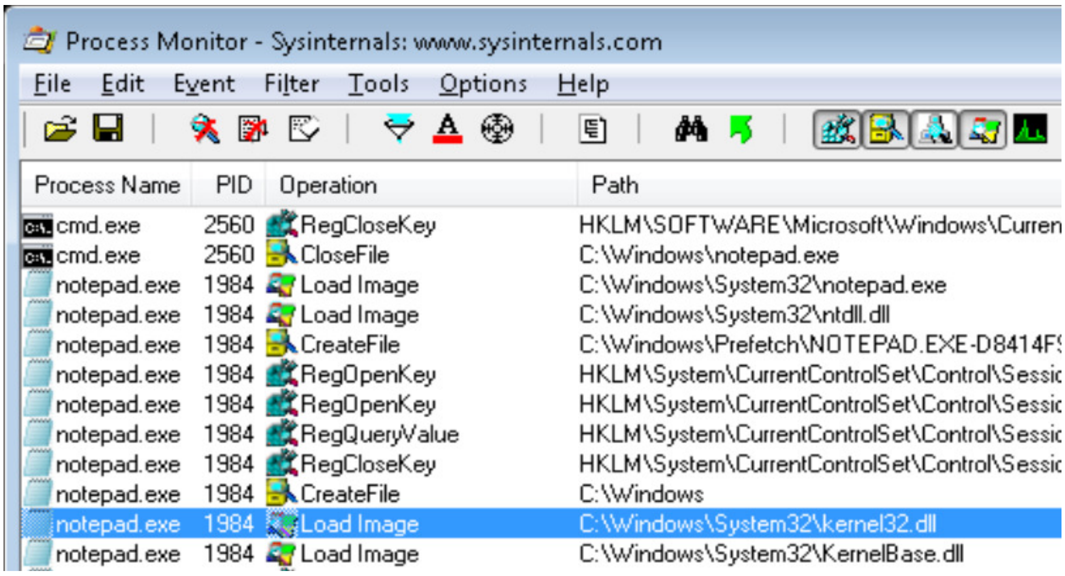
## 3.2 Monitoring and Measurement of Processes

Process monitoring is a powerful approach which can help programmers to diagnose and fix software issues, since it provides the capability to observe processes at a very low-level. Additionally, process monitoring can be used for testing purposes, such as tracing file system events induced by individual processes, e.g. web browsers.
Within the next sections process monitoring on Windows platforms is explained by giving an overview of Procmon's features which are essential for the new investigation approach.

### 3.2.1 Monitoring File System Events

As stated in [84], Windows processes are containers which provide resources for executing program code. Typically, a Windows process consists of a unique process identifier (PID), a parent process identifier (PPID), a list of threads, a virtual address space, an executable program, a list of process handles (e.g. semaphores, ports) and a security context.
Process monitoring in Windows can be accomplished with Procmon. This utility is capable of monitoring different aspects of a process like registry operations, file system operations, network

activity, (sub-)process creation or threading. Although Procmon offers several possibilities of monitoring different aspects of a process, this thesis aims to monitor file system events which have been induced by web browsers. Monitoring network or registry aspects of processes goes beyond the boundaries of this work, therefore they are omitted and may be part of a future work [84, 85].
Figure 3.1 illustrates Procmon's graphical user interface. As seen, the processes notepad.exe as



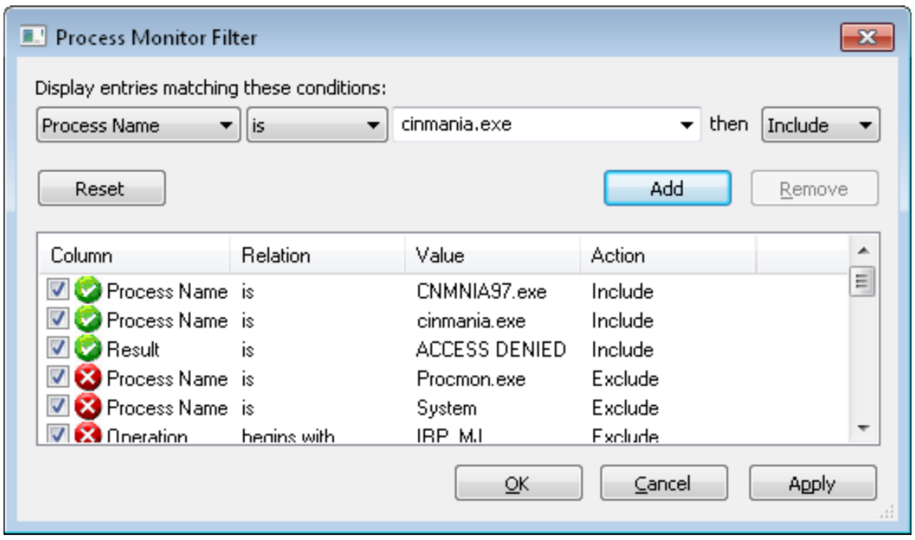
| Process Name | PID | Operation | Path |
|---|---|---|---|
| cmd.exe | 2560 | RegCloseKey | HKLM\SOFTWARE\Microsoft\Windows\Curren |
| cmd.exe | 2560 | CloseFile | C:\Windows\notepad.exe |
| notepad.exe | 1984 | Load Image | C:\Windows\System32\notepad.exe |
| notepad.exe | 1984 | Load Image | C:\Windows\System32\ntdll.dll |
| notepad.exe | 1984 | CreateFile | C:\Windows\Prefetch\NOTEPAD.EXE-D8414F |
| notepad.exe | 1984 | RegOpenKey | HKLM\System\CurrentControlSet\Control\Sessic |
| notepad.exe | 1984 | RegOpenKey | HKLM\System\CurrentControlSet\Control\Sessic |
| notepad.exe | 1984 | RegQueryValue | HKLM\System\CurrentControlSet\Control\Sessic |
| notepad.exe | 1984 | RegCloseKey | HKLM\System\CurrentControlSet\Control\Sessic |
| notepad.exe | 1984 | CreateFile | C:\Windows |
| notepad.exe | 1984 | Load Image | C:\Windows\System32\kernel32.dll |
| notepad.exe | 1984 | Load Image | C:\Windows\System32\KernelBase.dll |

**Figure 3.1:** Sample view of the main window of Microsoft's process monitoring utility Procmon [85]

well as cmd.exe are monitored. Furthermore, the GUI shows in column Operation the different types of operations a process has executed, such as CreateFile, which indicates an operation related to the file system [85].
According to [7], Microsoft has published a list of various file management functions which Procmon is able to monitor, e.g. OpenFile, CopyFile, DeleteFile.

### 3.2.2 Filtering and Logging File System Events

When using Procmon, important file system events can be separated from unimportant ones by the usage of filters. Procmon provides a GUI for defining filters, as shown in figure 3.2. Procmon's filter functionality can be used to restrict the processes which are going to be monitored as well as specific system events, in order to refine the monitoring results. After a filter has been defined, it is possible to export the filter to use it a later date. For example, by reusing the same filter for different processes, a comparison can be made between these processes (e.g. web browsers), thereby yielding in more accurate results.
Furthermore, Procmon provides three different file formats to save monitoring results: PML, XML and CSV. PML is Procmon's proprietary format and comes with the full quality of the



Process Monitor Filter

Display entries matching these conditions:

Process Name | is | cinmania.exe | then | Include

| Column | Relation | Value | Action |
|---|---|---|---|
| ✓ Process Name | is | CNMNIA97.exe | Include |
| ✓ Process Name | is | cinmania.exe | Include |
| ✓ Result | is | ACCESS DENIED | Include |
| ✓ Process Name | is | Procmon.exe | Exclude |
| ✓ Process Name | is | System | Exclude |
| ✓ Operation | begins with | IRP_MJ | Exclude |

**Figure 3.2:** Sample view of the filter settings menu of Microsoft's process monitor utility Procmon [84]

monitoring results. PML allows Procmon to import monitoring results on any system [84]. For the purpose of this thesis the CSV file format is chosen, since both the XML and the PML structure are more complex and consume too much disk space, whereas the structure of the CSV file format is simple and can be processed more effectively.

## 3.3 Monitoring the Private Browsing Mode of Modern Web Browsers

The following section presents a new approach for the digital investigation of the private mode of modern web browsers. This approach takes advantage of the VirtualBox and Procmon features which have been described before.

### 3.3.1 Conceptual Design of the Forensic Investigation Approach

The basic idea of monitoring modern web browsers is explained within this section. Figure 3.3 depicts a diagram consisting of two axis *b* and *t*, where *b* indicates, by how many percent the web browser has been loaded into the RAM for execution. Zero percent means that the virtual machine is up and running. A hundred percent means that the browser has been launched successfully in private mode, which is indicated by $x$. The variable *t* represents the time that has passed since the web browser start-up. The variables *y* and *z* represent in each case the state of the virtual machine, i.e. $y$ indicates that the web browser in private mode is being shut down and $z$ indicates that the web browser is shutdown as well as the virtual machine.
From a testing perspective it is assumed that the private mode of modern web browsers generates events (digital evidence) on the local hard drive between the web browser start-up ($x$) and

shutdown ($z$). This events are logged by means of Procmon.
From a conceptual point of view, an initial snapshot is created before the virtual machine is launched. Afterwards the virtual environment is started. Procmon is launched and configured with specific filter settings and started in capturing mode before the web browser is launched for testing, in order to log that file system events which have been triggered by the web browser. Then the web browser is launched in private mode for testing ($x$). During this time virtual machine screenshots are taken of each web site called in the web browser. The screenshots are stored to a shared virtual machine folder. After a specified time (y), testing is stopped and the web browser is shut down. In the meanwhile the process monitor has logged all file system events in form of a protocol which is exported in CSV format. The Procmon log files are saved to the shared virtual machine folder as well.

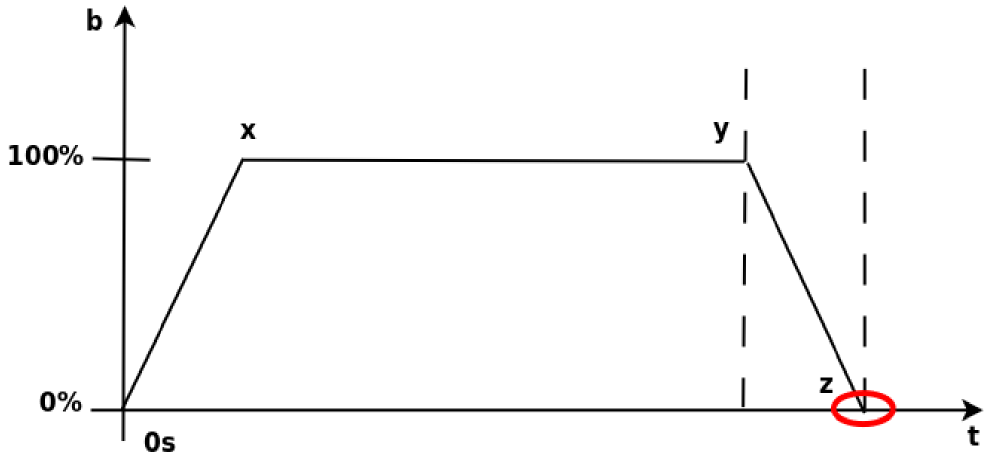


**Figure 3.3:** Description of the forensic analysis concept for investigating the private mode of modern web browsers

Next to that ($z$), the process monitor log file is cleared and the web browser is shut down as well as the virtual machine. After that a virtual machine snapshot is created which represents the state of the virtual machine hard disk at time $z$. Next, the virtual machine is reset to the initial state being ready for new test runs.
Finally, the initial snapshot and snapshot $z$ are imaged to two separate disk image files. Both image files are correlated with the Procmon log files that are contained on the virtual machine share. Afterwards the information gained from the previous correlation step is used to compare both snapshots to each other, in order to find and recover that digital evidence which has been left on the disk image after the web browser shut-down. A detailed technical description of this conceptual design is provided in the next chapter.

CHAPTER 4

# Implementation

This chapter presents the implementation of the testing and forensic investigation approach. First of all a holistic solution has been implemented, where both the testing and the analysis component have been automated (requiring minimal manual interaction). This implementation has been refused due to technical reasons regarding the process control of VirtualBox. The main problem was that Procmon could not be launched properly. In other words, the Procmon log files, which contain the monitored file system events, could not be created when Procmon was launched via the VirtualBox process creation API. Therefore, the implementation of the proposed approach was split into two automated parts that require more manual interaction than the prior solution.
One part is the testing framework, which is located on the guest system (virtual machine), and the other part is the analysis framework, which is situated on the host system. In this chapter, the configuration of both the testing and analysis environment is described. The implementation of both frameworks is presented, as well as the whole testing and analysis procedure.
The testing framework has been implemented in Java by means of Selenium [12], which is a web browser automation framework. The analysis framework has been implemented in The Python Programming Language (Python) and takes advantage of The Sleuth Kit [14], a forensic analysis tool kit, the fiwalk utility [53], which is part of The Sleuth Kit, and Digital Forensics XML (DFXML) files [53].

## 4.1 Configuration of the Test Environment

The configuration of the testing environment consists of the latest web browsers that have been available at test time. The testing environment is configured as follows:

- VirtualBox 4.3.10 r93012
- Windows 7 (Service Pack 1)
- Procmon 3.05

- Internet Explorer 11.0.9600.16428IS
- Firefox 28.0
- Chrome 33.0.1750.154 m
- Safari 5.1.7 (7534.57.2)
- Opera 20.0.1387.91
- Java Software Development Kit 1.7 Update 45
- Selenium Development Kit 2.39.0
- Selenium Web Driver for Internet Explorer 2.39.0
- Selenium Web Driver for Chrome 2.39.0
- ~20 GB Hard disk
- ~5,2 GB RAM

VirtualBox has been chosen to be the virtual environment. Windows 7 (Service Pack 1) has been selected to be the guest operating system, which has been set up on the virtual machine. Procmon has been chosen as process monitor utility. The web browsers Internet Explorer, Chrome, Firefox, Opera and Safari have been selected for testing purposes, since they are very popular due to their large user base [2].

### 4.1.1 Shared Folder Settings

As aforementioned in section 3.1.2, VirtualBox supports shared folders by means of the ”Guest Additions” [ [17], Oracle, 2013] feature. The testing framework uses this feature for sharing data between the guest and the host system. A shared folder, named test_share, is required by the testing framework. This folder is mounted as read/write on the guest system and contains two sub folders as follows:

- procmon_configs

  This folder contains several configuration files for Procmon. Generally, each configuration file can consist of different filter settings which are used for filtering file system events.

- test_result

  Within this folder the testing results are stored, i.e. the folder holds the generated Procmon log files, the URLs of the web sites which have been used for testing, as well as the screenshots of the visited web sites. The screenshots have been created by using VirtualBox's screenshot feature, as presented in section 3.1.3.

### 4.1.2 Process Monitor Filter Settings

Procmon has been set up to use a previously created configuration (PMC) file [84] that is located on the shared folder procmon_configs. Generally, Procmon event filters can be created and added to a Prcomon configuration file. These filters specify what system events are kept or dropped at monitoring time. For the purposes of this work, Procmon has been configured to include (keep) the file system events that match the conditions which are shown schematically in table 4.1. These events have been found to be suitable for forensic analysis, since they represent the well known file system operations: create file, modify file, delete file and rename file. As seen in the table, the filter settings additionally contain the process names of the web browsers which have been selected for testing.

| **Column** | **Relation** | **Value** | **Action** |
|---|---|---|---|
| Process Name | is | iexplore.exe | Include |
| Process Name | is | chrome.exe | Include |
| Process Name | is | firefox.exe | Include |
| Process Name | is | safari.exe | Include |
| Process Name | is | opera.exe | Include |
| Operation | is | CreateFile | Include |
| Operation | is | WriteFile | Include |
| Operation | is | SetDispositionInformationFile | Include |
| Operation | is | SetRenameInformationFile | Include |

**Table 4.1:** Schematic view of Microsoft Procmon's include filters used in the implementation of the testing framework [7, 16, 84]

According to [7, 16], the file system operations from table 4.1 can be interpreted as follows: CreateFile is registered when a file is created or opened for reading. WriteFile is registered when data is written to a file. SetDispositionInformationFile is registered when a file is marked for deletion. Usually, such files are deleted after they have been closed, as initial testing has shown. SetRenameInformationFile is registered when a file is renamed.

Additionally, Procmon has been configured to exclude (drop) all events which suit to the conditions, as presented in table 4.2.

| **Column** | **Relation** | **Value** | **Action** |
|---|---|---|---|
| Process Name | is | Procmon.exe | Exclude |
| Path | ends with | .icm | Exclude |
| Path | ends with | .ICM | Exclude |
| Path | ends with | .exe | Exclude |
| Path | ends with | .EXE | Exclude |
| Path | ends with | .ttf | Exclude |
| Path | ends with | .TTF | Exclude |
| Path | ends with | .ttc | Exclude |
| Path | ends with | .TTC | Exclude |
| Path | ends with | .dll.mui | Exclude |
| Path | ends with | .DLL.MUI | Exclude |
| Path | ends with | .pf | Exclude |
| Path | ends with | .PF | Exclude |
| Path | ends with | .nls | Exclude |
| Path | ends with | .NLS | Exclude |
| Path | ends with | .dll | Exclude |
| Path | ends with | .DLL | Exclude |
| Path | ends with | .tlb | Exclude |
| Path | ends with | .TLB | Exclude |
| Path | ends with | .customDestinations | Exclude |
| Path | ends with | .customDestinations-ns | Exclude |
| Path | ends with | .ini | Exclude |
| Path | ends with | .INI | Exclude |
| Path | ends with | .Manifest | Exclude |
| EventClass | is | .Registry | Exclude |
| EventClass | is | .Network | Exclude |
| EventClass | is | .Process | Exclude |
| EventClass | is | .Profiling | Exclude |

**Table 4.2:** Schematic view of Microsoft Procmon's exclude filters used in the implementation of the testing framework [84]

The filters in table 4.2 pre-filter files which can be classified as noise generated by the operating system. The digital evidence contained in these files is negligible for further analysis, as initial tests have shown. Therefore, these files are omitted. Additionally, the system events which belong to the event classes .Registry, .Network, .Process and .Profiling are excluded from logging, as the focus of this Master's thesis lies on monitoring file system events [84].

### 4.1.3 Web Browser Settings

The configuration of the Internet Explorer, Chrome, Firefox, Opera and Safari has remained unchanged, except some minor modifications. Due to performance issues that have been observed during initial testing, the web browsers have been configured to block pop-ups.
Add-ons and plug-ins have been disabled, which has been found to be useful at test time. Initial tests also have shown that enabled web browser add-ons and plug-ins slow down the testing process. Therefore, these features have been disabled as well.
The set-default-browser option has been disabled for the selected web browsers, otherwise every time on web browser start-up a notification is displayed by forcing the user to set a default web browser. The browsing history has been cleared and disabled by default for each web browser.

## 4.2 Description of the Testing Framework

As mentioned before, the testing framework is written in Java and takes advantage of Selenium's web browser automation features. The following sections describe the overall testing procedure in form of a UML sequence diagram, as seen on figure 4.1. Due to technical limitations that are explained in section 4.5.1, applying the automated testing procedure has been found to be suitable only for the Internet Explorer and Chrome. Therefore, the descriptions provided in sections 4.2.2 and 4.2.3 are focusing on the automation of these web browsers. However, Firefox, Safari and Opera have been tested according to the steps of the overall testing procedure in a way that the automated steps have been performed manually, as presented in chapter 5.

### 4.2.1 Definition of the Overall Testing Procedure

The overall testing concept, which has been implemented, is illustrated in figure 4.1.

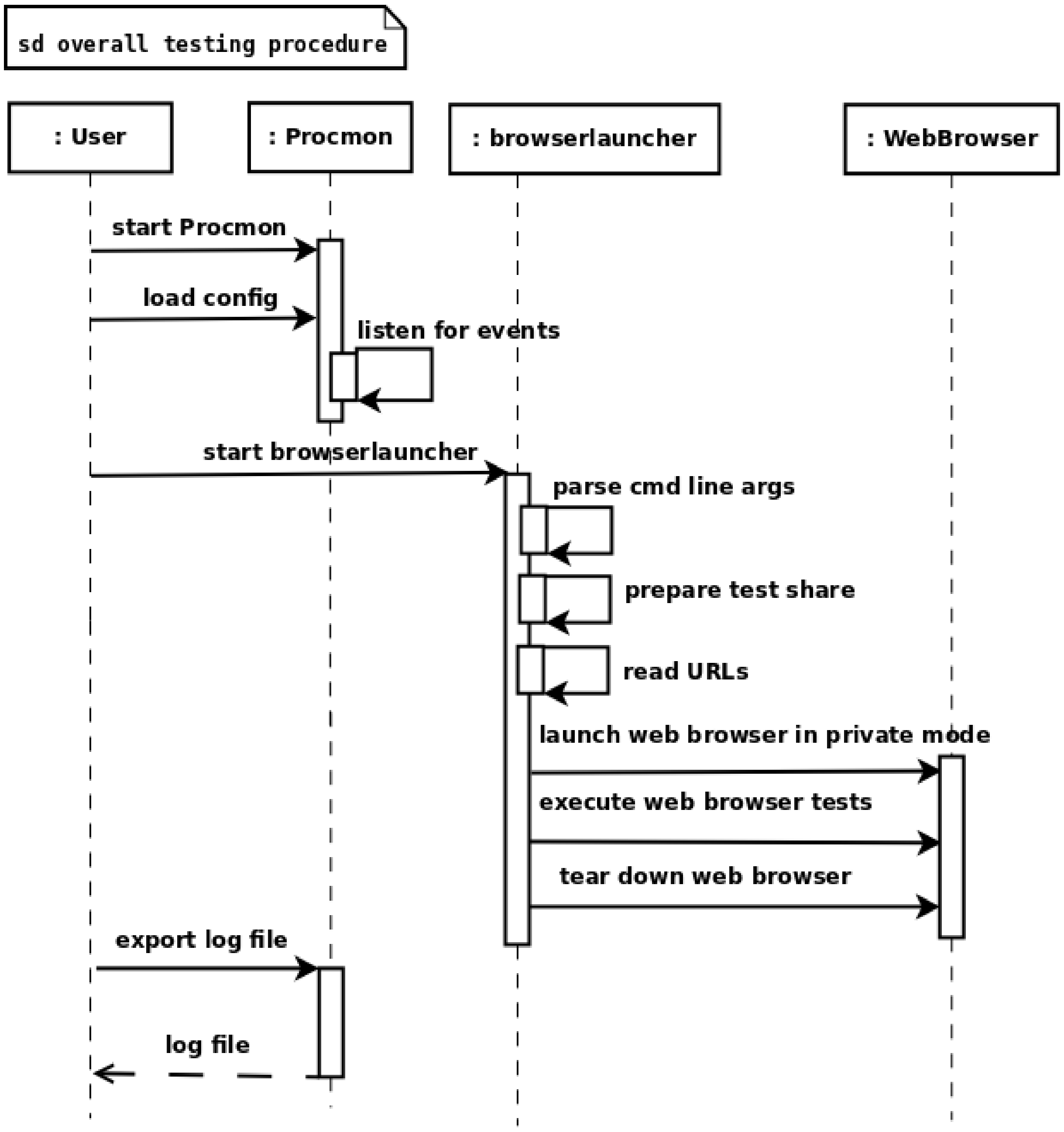


**Figure 4.1:** Description of the overall testing procedure for testing web browsers in private mode

The testing steps from figure 4.1 are explained as follows:

1. start Procmon

   The testing environment is started and Procmon is launched.

2. load config

   Procmon is provided with a configuration file that is located on the shared folder /test_share/procmon_configs. This file contains the settings that are necessary for filtering those file system events which have been induced by a specific web browser, e.g. the Internet Explorer, Chrome.

3. listen for events

   After Procmon has been configured properly, it starts capturing file system events which match the specified filter criteria.

4. start browserlauncher

   The browserlauncher application is executed with the command line arguments provided by the user. This Java application represents the core of the testing framework which is used for web browser automation and testing purposes.

5. parse cmd line args

   The browserlauncher parses the provided command line arguments and determines the web browser which needs to be started in private mode.

6. prepare test share

   The virtual machine shared folder from section 4.1.1 is prepared by the browserlauncher, i.e. for each test run a sub folder is created under /test_share/test_result. This sub folder contains the virtual machine screenshots and a browserlauncher application log file consisting of the web sites that timed out. The screenshots and application log files are not part of the experimental evaluation, as they have been only used for diagnostic purposes during testing. Furthermore, the Procmon log files are stored to that sub folder for further analysis.

7. read URLs

   The browserlauncher parses the URLs which have been provided via input file from /test_share/test_result.

8. launch web browser in private mode

The browserlauncher starts that web browser in private mode which has been provided via command line argument. Section 4.2.2 explains the features which are part of this step in more detail.

9. execute web browser tests

   The browserlauncher calls the URLs that have been parsed from the command line in the previously launched web browser. A more technical description of the sub steps that are part of this step is provided in section 4.2.3.

10. tear down web browser

    After testing has been finished the browserlauncher quits the web browser and shuts down.

11. export log file

    After testing has been finished, the Procmon log files need to be exported manually to that shared folder, which has been created for storing the test results,
    e.g. /test_share/test_result/ie_test_result_20140501.

The following sections cover technical aspects of the Selenium configuration that has been used in conjunction with the automation of Internet Explorer's and Chrome's private mode. Additionally, the step "execute web browser tests" (step nine) from the overall testing procedure is explained in more detail.

### 4.2.2 Launching Web Browsers in Private Mode

The testing framework uses Selenium for web browser automation. Selenium provides a component, called WebDriver, [13] which is a generic Java class for launching web browsers. When launching the Internet Explorer or Chrome via Selenium, the implementing classes of WebDriver, which are the InternetExplorerDriver [10] and the ChromeDriver [4], have been used. According to [5], the InternetExplorerDriver has been configured to use the following properties at start-up:

| **Property** | **Value** |
|---|---|
| webdriver.ie.driver | path_to_driver\IEDriverServer.exe |
| ie.forceCreateProcessApi | true |
| ie.browserCommandLineSwitches | -private |
| ie.ensureCleanSession | true |

**Table 4.3:** Schematic view of the settings required by the Selenium framework for launching Internet Explorer's private mode [5]

As seen in table 4.3, the InternetExplorerDriver requires the path to the binary IEDriverServer.exe to be set, in order to launch the Internet Explorer. Furthermore, Selenium requires the Windows

Registry key Computer\HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main \TabProcGrowth to be set with the value zero and the ie.* options to be configured, so that the Internet Explorer can be launched in private mode [10].
According to [3–5], the ChromeDriver uses the following properties at start-up:

| **Property** | **Value** |
|---|---|
| webdriver.chrome.driver | path_to_driver\chromedriver.exe |
| --user-data-dir | path_to_chrome_user_profile\Google\Chrome\User Data |
| --incognito | true |

**Table 4.4:** Schematic view of the settings required by the Selenium framework for launching Chrome's private mode [3–5]

As seen in table 4.4, the ChromeDriver uses the path to the binary chromedriver.exe [4]. The other options from table 4.4 are used to launch Chrome's private profile.

### 4.2.3 Testing Web Browsers in Private Mode

When performing web tests, the browserlauncher application controls the browser under test. According to figure 4.2, the browserlauncher carries out the following steps for each URL that is tested:

1. navigate to URL

   The browserlauncher is built upon the "Implicit Wait" [ [15], Selenium, 2013 ] component of Selenium. This feature is used when navigating between arbitrary URLs. After a specified time the "Implicit Wait" [ [15], Selenium, 2013 ] feature notifies the browser to call the next URL in the address bar.

2. log to time out log file

   If a specific web site is not loaded within the specified period of time, the browserlauncher logs the name of the web site to a log file that has been created on the virtual machine shared folder, e.g. /test_share/test_result.

3. take screenshot

   Finally, the browserlauncher takes a screenshot of each web page regardless of whether or not the web page has been loaded completely.

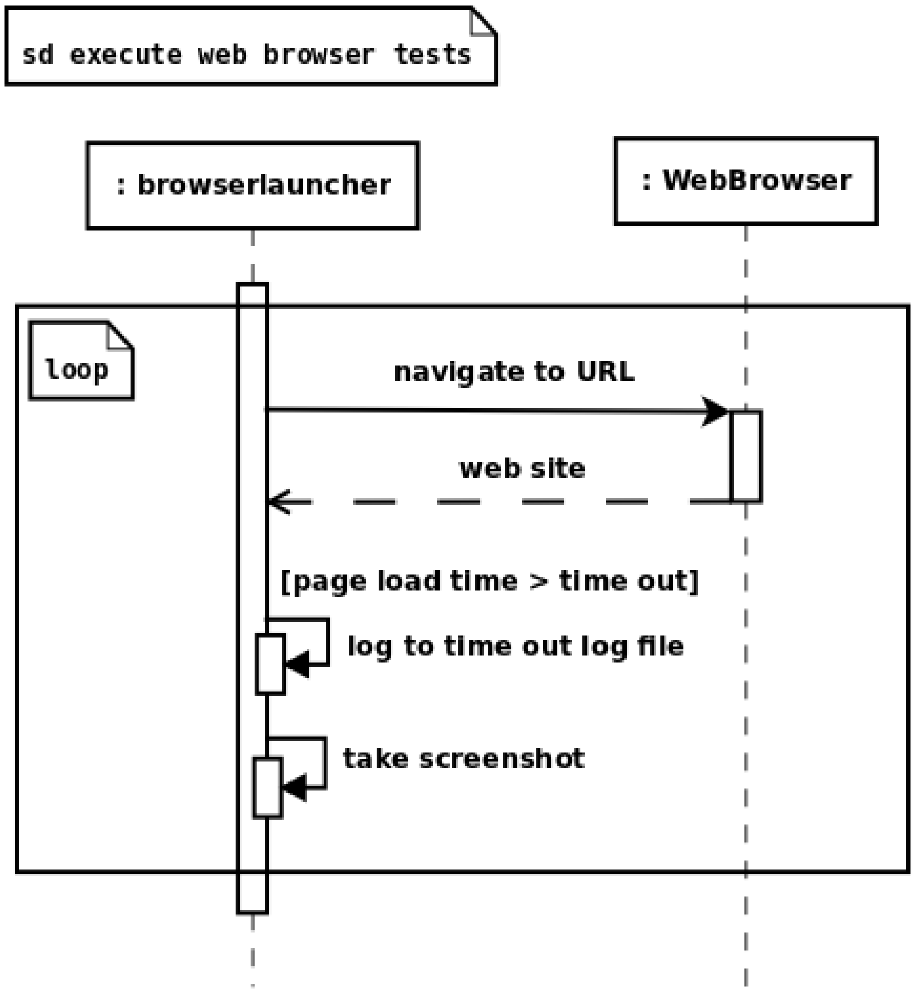


**Figure 4.2:** Detailed description of the step ”execute web browser tests” from the overall testing procedure

## 4.3 Configuration of the Analysis Environment

The configuration of the analysis environment differs form the configuration of the testing environment. The analysis environment has the following characteristics:

- Python 2.7
- The Sleuth Kit 4.1.2
- DFXML Python binding 1.0.1
- Fiwalk Python binding 0.6.3
- ~500 GB Hard disk
- ~8 GB RAM

Python has been used for imaging the VirtualBox snapshots to disk image files. The Sleuth Kit release contains the fiwalk utility which can be used to process disk images efficiently. Additionally, fiwalk is able to create DFXML files when processing disk images. Generally, a DFXML file represents a file object, which is residing on a disk image, in XML structure [53]. Both the DFXML and the fiwalk Pyhton binding are required for producing DFXMLs.
The overall analysis procedure is described more detailed in section 4.4, but before the configuration of the analysis framework is presented.

### 4.3.1 Storage Controller Settings

VirtualBox maintains a XML file (.vbox) for each virtual machine [17]. A .vbox file holds different configuration data, such as snapshot UUIDs that are mapped to one or more storage controllers. A storage controller represents the ”hard disk” of a virtual machine. As seen in listing 4.1, the snapshot with the UUID ”7a0467d2-d0d7-43dd-9f5f-7177d3b46e10” has attached one SATA storage controller. The analysis framework requires only one storage controller per snapshot to be configured. Each storage controller represents the state of a particular snapshot at a specific time, for example, the state of the hard disk after a web browser test, which can be used for imaging.

```
<Snapshot uuid="{7a0467d2-d0d7-43dd-9f5f-7177d3b46e10}">
 ...
 <StorageControllers>
   <StorageController name="SATA">
     <AttachedDevice type="HardDisk">
       <Image uuid="{1386b43a-4e6f-42b4-adf0-84313bce3e94}"/>
     </AttachedDevice>
   </StorageController>
 </StorageControllers>
<Snapshot>
```

**Listing 4.1:** Schematic view of the storage controller section of VirtualBox’s .vbox XML file [17]

Other VirtualBox options like host I/O caching, which is used to cache the write operations of the virtual hard disk, or the property ”setextradata” [ [17], Oracle, 2013 ], which can be used to set the commit rate of the guest file system to the virtual hard disk, have remained unchanged, since disk imaging as well as the forensic analysis take place after the virtual machine shutdown [17]. This virtual machine configuration is considered to be forensically sound as well as comparable to a real computer system, because it is assumed that all cached write operations have been written to the storage controller after the virtual machine shutdown.

### 4.3.2 Folder Structure of the Analysis Framework

The analysis framework uses the following folder structure:

- analysis_folder

  This folder contains the init folder and one or more snapshot folders, which are used for further analysis.

- init_folder

  This folder contains the disk image of the initial snapshot and its corresponding DFXML file.

- snapshot_folder

  Beneath a snapshot folder there are two sub folders, called analysis_result and test_result. The analysis_result folder contains the disk image of a particular snapshot, its corresponding DFXML file and the recovered browsing data, such as cookies or graphics files. The test_result folder consists of the log files generated by Procmon as well as the virtual machine screenshots, the time out log file and the tested URLs.

A detailed explanation of how Procmon log files, disk images and DFXML files are used when analysing disk images is provided in the following sections.

## 4.4 Description of the Forensic Analysis Framework

The analysis framework has been written in Python and uses features of The Sleuth Kit, the fiwalk utility and DFXML. Similar to the testing framework, the following section describes the overall analysis procedure in form of a UML sequence diagram, as seen in figure 4.3. Additionally, particular sections of the analysis procedure are described in more detail for a better understanding.

### 4.4.1 Definition of the Overall Analysis Procedure

The analysis framework implements the concept of comparing virtual machine snapshots to process monitor log files, in order to find digital evidence which has been created in the context

of a private browsing session. According to section 3.3.1, at least two snapshots have to be supplied to the analysis framework, where one snapshot is referring to the state of the virtual hard disk before a web browser test (initial snapshot) and at least one snapshot is referring to the virtual disk state after a web browser test (snapshot X). By means of the analysis framework, it is possible to compare the initial (”clean”) snapshot to a snapshot X that contains private browsing data.
The overall analysis procedure is depicted in figure 4.3.

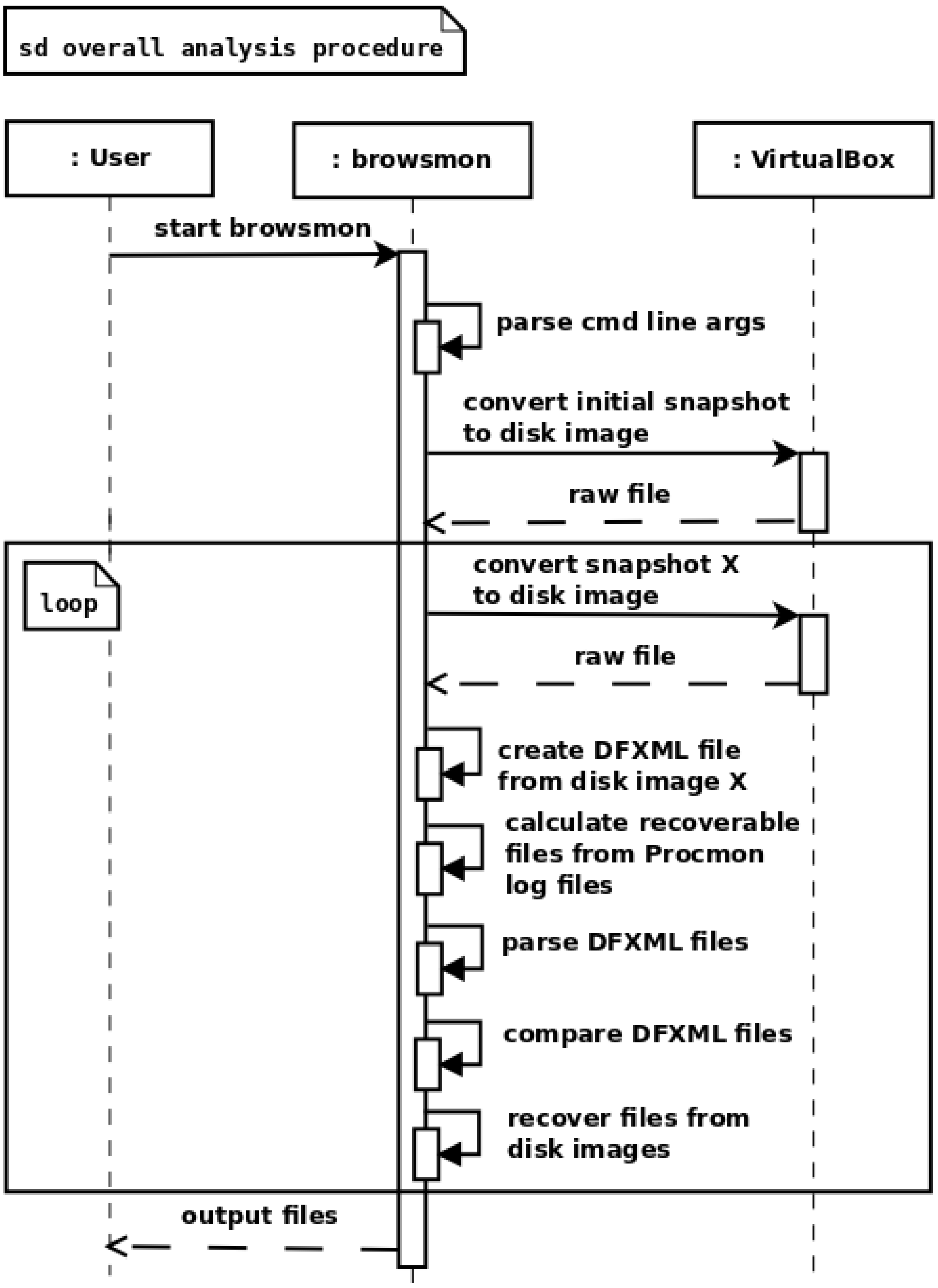

sd overall analysis procedure
: User
: browsmon
: VirtualBox
start browsmon
parse cmd line args
convert initial snapshot to disk image
raw file
loop
convert snapshot X to disk image
raw file
create DFXML file from disk image X
calculate recoverable files from Procmon log files
parse DFXML files
compare DFXML files
recover files from disk images
output files

**Figure 4.3:** Description of the overall forensic analysis procedure for examining the private mode of modern web browsers

According to figure 4.3, the following steps are performed when conducting a forensic analysis of the private mode of modern web browsers:

1. start browsmon

   The analysis framework consists of the Python script browsmon.py (browsmon) which is used for performing the forensic analysis of different virtual machine snapshots.

2. parse cmd line args

   Browsmon parses the provided command line arguments to determine the virtual machine and the corresponding snapshots for further analysis.

3. convert initial snapshot to disk image

   The browsmon application requests VirtualBox to convert the virtual hard disk of the initial snapshot to a disk image. Additionally, a DFXML file is generated based on that disk image.

4. convert snapshot X to disk image

   This step and the following ones are carried out iteratively for each provided snapshot. Browsmon converts the virtual hard disk of an arbitrary snapshot X to a disk image X.

5. create DFXML file from disk image X

   Browsmon creates a DFXML file from disk image X. A detailed explanation of creating such XML files is given in section 4.4.2.

6. calculate recoverable files from Procmon log files

   Browsmon parses the Procmon log files which are related to the snapshot X and calculates all potentially recoverable files. Within this step duplicate file names are filtered, due to the fact that a file can be contained multiple times within a Procmon log file.

7. parse DFXML files

   Generally, the analysis framework uses DFXML files to map file paths to inode numbers. The list of potentially recoverable files contains the relative file paths of those files which have been logged during a particular private browsing session. Browsmon takes each file path and searches for its corresponding inode number within the initial snapshot DFXML file as well as within the snapshot X DFXML file. For each valid inode number a file object is extracted from the aforementioned DFXML files (initial DFXML and/or snapshot X DFXML). Afterwards, these file objects are stored to separate XML files. A more detailed description of creating and extracting DFXML file objects, is presented in section 4.4.2.

8. compare DFXML files

   The DFXMLs of matching file objects (those with the same file path) are compared to each other. For matching files a .diff file is created which stores the result of the comparison. Section 4.4.3 covers the process of comparing DFXML file objects in more detail.

9. recover files from disk images

   The file objects with a valid inode number are recovered from the initial disk image and/or the snapshot X disk image for further analysis purposes. In other words, by taking advantage of inode numbers, web browser artefacts, such as the browsing history or cookies, can be recovered. This step is described in more detail in section 4.4.4.

### 4.4.2 Creating Digital Forensics XML Files

In [52, 53], Garfinkel presents the fiwalk utility as part of The Sleuth Kit framework. As mentioned before, fiwalk is capable of extracting file objects as DFXML files from arbitrary disk images. Listing 4.2 shows a sample of a DFXML file object, called fwlink[1].htm, which has been located in the Internet Explorer's web cache directory.

```
<fileobject>
   ...
   <filename>Users/win7/AppData/Local/Microsoft/Windows/
           Temporary Internet Files/Content.IE5/U92U03D9/fwlink[1].htm
   </filename>
   ...
   <name_type>r</name_type>
   <filesize>182</filesize>
   ...
   <inode>60911</inode>
   ...
   <mtime>2013-11-14T13:59:33Z</mtime>
   <ctime>2013-11-14T13:59:33Z</ctime>
   <atime>2013-11-14T13:59:33Z</atime>
   <crtime>2013-11-14T13:59:33Z</crtime>
   ...
   <byte_runs>
    <byte_run file_offset='0' fs_offset='0' img_offset='105906176'
             len='182' type='resident'/>
   </byte_runs>
   <hashdigest type='md5'>8a11c6169f03fc4d310fc8c0f947cb54</hashdigest>
   <hashdigest type='sha1'>6ca3cd1e41678833dc1ac45a71a78276ca5e8585
   </hashdigest>
</fileobject>
```

**Listing 4.2:** Schematic view of a file object in DFXML format produced by the fiwalk utility [53]

A detailed description of the DFXML tags is available at [6]. The XML tags listed listing 4.2 are used for further analysis, since they provide valuable information to investigators, such as the file name, MAC times or the information whether or not a file's content has changed. The XML tags from listing 4.2 are describes as follows [6]:

- filename

  This attribute describes the full path of the file relative to the root folder, i.e. C: or D: in Windows.

- name_type

  This attribute identifies the file type. For example, r stands for regular file. Also directories, hard links, named pipes, etc. are valid file types.

- filesize

  This attribute reports the file size in bytes.

- inode

  This attribute reports the inode number of a file object.

- mtime, ctime, atime, crtime

  These attributes describe a file's MAC times as follows: mtime stands for last modification time, ctime stands for last modification of the meta data, atime stands for last access time and crtime stands for the creation time of a file.

- byte_runs

  This attribute reports the byte runs of a file which means that a file is organised in a byte array on the hard disk. Byte_runs contains the cryptographic hash value for each array element.

- hashdigest

  This attribute refers to the cryptographic hash value that is calculated from the whole file content. Typically, fiwalk calculates the md5 and sha1 hash sum of a file.

The remaining XML tags are omitted, since they are not necessarily required for the further evaluation, as presented in chapter 5.

### 4.4.3 Comparing Digital Forensics XML Files

After the file objects have been extracted from the initial snapshot DFXML file as well as form an arbitrary snapshot X DFXML file, the analysis framework creates a .diff file for matching file objects (file objects with the same file path), as seen in listing 4.3.

```
<fileobject>
   ...
   <filename>Users/win7/AppData/Local/Google/Chrome/User Data/
             Default/History-journal
   </filename>
   ...
   <name_type>r</name_type>
   <filesize>16384</filesize>
   ...
   <inode>66355</inode>
   ...
-  <mtime>2013-12-08T16:23:28Z</mtime>
?                   - - ^^^

+  <mtime>2013-12-08T22:31:35Z</mtime>
?                    ++++   ^

-  <ctime>2013-12-08T16:23:28Z</ctime>
?                    - - ^^^

+  <ctime>2013-12-08T22:31:35Z</ctime>
?                    ++++   ^

   <atime>2013-11-14T14:46:51Z</atime>
   <crtime>2013-11-14T14:46:51Z</crtime>
   ...
   <byte_runs>
     <byte_run file_offset="0" fs_offset="1812656128"
               img_offset="1918562304" len="16384"/>
   </byte_runs>
-  <hashdigest type="md5">6ce5b8b6b73e3149eea54d5e8e9f5f58</hashdigest>
-  <hashdigest type="sha1">01965d4158c06ee6e395aa32ee66bac1c7ba8988
   </hashdigest>
+  <hashdigest type="md5">50c518c6e2dae518aed504e231c6249b</hashdigest>
+  <hashdigest type="sha1">d030648ead25a7bfbb55511f76628f7fecbc77c3
   </hashdigest>
</fileobject>
```

**Listing 4.3:** Schematic view of a .diff file showing the differences between two DFXML file objects

The XML excerpt in listing 4.3 shows that the Chrome's History-journal file has changed. As can be seen, the file's mtime, ctime and both hash digests have been modified.

### 4.4.4 Recovering Data From Disk Images

The last step being performed by the analysis framework is the recovery of private browsing data. The current implementation uses the icat [8] utility, which is part of The Sleuth Kit, for file recovery purposes. Icat is run by the analysis framework via Python's sub process control mechanisms. File contents (no directories) are recovered according to their inode number, irrespective of their allocation status (e.g. allocated, unallocated). Ignoring a file's allocation status will enable investigators to retrieve more potentially valuable information about what has happened on the file system during a private browsing session. Slack space is not recovered, because the current solution focuses on deleted file recovery. Recovering slack space will require an approach that uses file carving techniques. This would exceed the borders of this work and therefore was not implemented.

## 4.5 Lessons Learned

This section summarises the technical challenges that were observed during the implementation phase.

### 4.5.1 Testing Framework

First of all it was challenging to get the web browsers started in private browsing mode properly, because each browser has special characteristics and requires different settings, i.e. Chrome uses browsing profiles, whereas the Internet Explorer does not use any profile.
It should be mentioned that the Selenium framework is evolving fast and it was observed that several versions of the framework were published while the testing framework was implemented. The problem was that some versions of the Selenium framework were not compatible with some web browser versions. One is advised to keep the Selenium framework up to date when testing new browser versions.
Another obstacle concerning the Selenium framework was that not all five web browser could be launched automatically, due to the limited support of Selenium's WebDriver class. For example, Opera's latest version was not supported by Selenium. Safari's private mode could not be launched automatically, due to problems concerning the command line parameter handling. Furthermore, the test data generated by Firefox could not be recovered from the virtual machine disk image, because the WebDriver class erases Firefox's temporary working directory permanently on web browser shutdown. Therefore, only a partially automated approach has been implemented by focusing on the Internet Explorer and Chrome as well.
It should be noted that the current implementation of Selenium is not able to handle more than one web browser tab at test time. Therefore, all web sites are called in the same browser tab sequentially.
Due to technical reasons of VirtualBox's process creation API, the initial testing and analysis approach was refused. While the initial approach was implemented, it was observed that Procmon crashed all the time, i.e. the configuration (PMC) file could not be applied to Procmon automatically. Furthermore, Procmon's log files could not be extended automatically during the initial testing phases.

In addition it has been found that web browser testing via Selenium is a time consuming task. Initial tests have shown that calling 1000 URLs (by using a time out of 10 seconds per URL) via the browserlauncher takes about three hours. It should be mentioned that Procmon generates several log files depending on the testing purposes. Each Procmon log file will contain at most 450 MB of log data depending on the amount of tested web sites. Investigators are advised to provide enough disk space for storing log files.

### 4.5.2 Analysis Framework

As the implementation of the analysis framework went on, it was observed that the forensic analysis of disk images was a time consuming task. An initial analysis has shown that converting VirtualBox storage controllers to disk images of approximately 20 GB in size takes about 30 to 40 minutes. Creating DFXML files from disk images by means of the fiwalk utility takes seven to ten minutes per disk image, as parsing those DFXML files lasts around three to five minutes per file. Comparing DFXML files as well as recovering file contents from a disk image is performed in a rather short amount of time.
Another technical detail to mention is that ifind [9] initially was used to find the inode information of potentially recoverable files. This mechanism was refused due to performance issues. Thus, searching inode numbers within DFXML files has been found to be a more appropriate solution, since testing has shown that parsing XML files can be performed more efficiently than calling ifind via Python's sub process control mechanisms.

CHAPTER 5

# Experimental Evaluation

In this chapter the experimental evaluation of the practical part of this work is presented. Firstly, three different testing scenarios are defined that have been selected for web browser testing. Secondly, the procedures for generating and collecting test data are described. Finally, the private browsing artefacts are presented, which have been found in the course of the digital investigation.

## 5.1 Definition of the Test Scenarios

Defining test cases is a crucial task when verifying the user privacy features of modern web browsers from a forensically point of view. The following test cases have been found to be valuable: The test scenario described in section 5.1.1 allows investigators to create a baseline of those Internet artefacts which are accessed during a public browsing session. This baseline can be compared to the test scenarios defined in section 5.1.2 and in section 5.1.3, which evaluate the Internet artefacts a web browser accesses in private mode.

### 5.1.1 Public Browsing Mode Launched Manually

By performing this testing scenario, the web browser is launched manually. A single browser window is used for testing, which is configured to be in public browsing mode. No browser tabs are used. During testing, the web sites are accessed in the same browser window consecutively. A new web site is called as soon as the previous site has been loaded.

### 5.1.2 Private Browsing Mode Launched Manually

This testing scenario is similar to the testing scenario presented in section 5.1.1, except that the browser under test is launched in private browsing mode.

### 5.1.3 Private Browsing Mode Launched Automatically

In addition to manual testing, the private browsing mode of a web browser is tested automatically by means of the testing framework, as explained in section 4.2.

## 5.2 Generating Test Data

All tested web sites have been selected according to Alexa's web site ranking [1]. The top three web sites (www.google.com, www.facebook.com and www.youtube.com) have been used for testing the public and private browsing mode of the Internet Explorer, Chrome, Firefox, Safari and Opera manually. As initial tests have been performed, this approach has been found to be suitable, as testing a small set of web sites generates a manageable amount of test data that can be evaluated, in order to find valuable Internet artefacts. This step has been performed for each of the five web browsers.
Within the next step Alexa's top 300 web sites have been selected for testing the private browsing mode of the Internet Explorer, Chrome, Firefox, Safari and Opera. This step has been performed manually, due to the technical issues regarding Selenium.
The last step deals with the automation of the Internet Explorer and Chrome, as both browsers offer mechanisms for launching their private mode automatically. For each web browser the top 1200 web sites of the Alexa ranking have been selected. Each set can be divided into four sub sets of 300 web sites. As initial tests have shown, a set of 300 web is sites suitable when testing a web browser's private mode.

## 5.3 Collecting Test Data

In accordance with the testing scenario from section 5.1.1, the top three web sites of the Alexa ranking have been called in each web browser subsequently. As a result, ten different virtual machine snapshots have been collected, i.e. for each web browser a snapshot has been created that has contained the public browsing data and another one that has contained the private browsing data. According to the testing scenario from section 5.1.2, the private mode of each web browser has been tested manually by using Alexa's top 300 web sites. This step has resulted in five additional snapshots. Conforming to the test scenario from section 5.1.3, the private mode of the Internet Explorer and that one of Chrome has been tested automatically by using Alexa's top 300 web sites, which has produced eight additional snapshots.

## 5.4 Test Results and Evaluation

Within this section an overview of the evaluation is given. Furthermore, the evaluation of the test results is presented in more detail.

### 5.4.1 Overview

In general, the experimental evaluation includes allocated and unallocated file objects that have been found in the virtual machine disk images. Data contained in slack space, orphaned data as well as overwritten data is omitted, since this would exceed the scope of this work.
Within the following sections there are presented two types of tables. The columns of table type one are interpreted as follows:

- Web Browser

  This column states the browser name.

- Files With Inodes

  This column displays the number of distinct file names which have been connected to a valid inode number (even after web browser shutdown), regardless whether the file content was recoverable or not.

- Snapshot X

  This column covers those files, which have been contained in an arbitrary snapshot X.

- Snapshot X / Initial Snapshot

  This column states which files have been contained in a specific snapshot X as well as in the corresponding initial snapshot. Basically this means that a file, for example, called $y$, has been located under a specific path in snapshot X and the same file name (plus file path) has been located in the initial snapshot. The matching criteria of both file objects is the file path. File $y$ from snapshot X and the corresponding file $y$ from the initial snapshot have been recoverable.

The columns of table type two are interpreted as follows:

- Internet Artefact

  This column states the name of the Internet artefact, which has been found in a particular disk image.

- Attribute

  This column presents the attributes of an artefact, which are: the file path pointing to that artefact, its MAC times as well as its SHA1 sum. From a forensically standpoint of view, these attributes can be considered to proof the existence of digital evidence in the context of an examination.

- Value

  This column presents the attribute values of a file's path, MAC times and SHA1 sum.

In this work, table type two is used to highlight the findings, which have been recovered from different private browsing sessions, whereas table type one states the number of recovered files in general. When describing public browsing artefacts, table type two is omitted, since this work focuses on private browsing artefacts. The public browsing artefacts in section 5.4.2 are only described for a better understanding of the entire work. The private browsing artefacts in section 5.4.2 are presented by using table type two.
Generally, private browsing artefacts have been accessed locally during each test run. Each web browser starts writing and / or modifying and / or removing private browsing data on start-up, during the browsing session and on shutdown, which has been observed when analysing the collected Procmon log files.
During the analysis phase a big amount of private browsing artefacts of the Internet Explorer has been exposed. Many different file types have been detected. The following sections present a representative for each valuable file type in more detail by using table type two. These representatives cover digital evidence and can be used to create a chain of custody in the course of a digital investigation. Additionally, the sections present those web sites (out of the tested ones) the private browsing artefacts have been referring to.
Contrary to the Internet Explorer, the other web browsers have not revealed valuable private browsing data, except Safari. The valuable private browsing artefacts of Safari are described by the use of table type two as well. The presentation of the private browsing artefacts created or modified by the other web browsers is omitted, as no digital evidence has been exposed regarding the visited web sites.

### 5.4.2 Evaluation of the Manual Test Scenarios

This section presents the evaluation of the Internet artefacts which have been generated by the public and private mode of the Internet Explorer, Firefox, Chrome, Safari and Opera in course of the manual testing phase. A comparison of these artefacts is provided in order to point out potential similarities and differences.

#### Public Browsing Artefacts

Table 5.1 shows the number of browsing artefacts that have been accessed when testing the public browsing mode of the Internet Explorer, Chrome, Firefox, Safari and Opera by means of Alexa's top three web sites www.google.com, www.facebook.com and www.youtube.com.

| Web Browser | Files With Inodes | Snapshot X | Snapshot X / Initial Snapshot |
|---|---|---|---|
| Internet Explorer | 165 | 131 | 22 |
| Chrome | 204 | 63 | 63 |
| Firefox | 91 | 83 | 2 |
| Safari | 52 | 46 | 0 |
| Opera | 160 | 134 | 12 |

**Table 5.1:** Public browsing artefacts generated by the Internet Explorer, Chrome, Firefox, Safari and Opera when tested Alexa's top three web sites manually

***Internet Explorer*** As can bee seen in table 5.1, Procmon has monitored 165 distinct files which the Internet Explorer has accessed in public mode. 154 files have been recovered from the virtual machine disk images, whereof 131 files have been contained in snapshot X and 22 files have been contained in both snapshots.
As the investigation took place, many Javascript files, XML files and graphics files, such as PNG or JPEG, have been discovered. But also other file objects that have no well known file name extensions. For example, the file 4309200C3DBAD0F6F0DFACE9165FD092 located under path /Users/pc/AppData/LocalLow/Microsoft/CryptnetUrlCache/MetaData does not have any extension. This file has consisted of binary data, as the analysis has revealed. Several cookies from www.google.com, www.facebook.com and www.youtube.com have been found under /Users/pc/AppData/Roaming/Microsoft/Windows/Cookies. For example, a binary file, called {AA052660-CCC6-11E3-9F3D-080027776A60}.dat, under path /Users/pc/AppData/Local/Microsoft/Internet Explorer/Recovery/High/Active has contained the URLs of www.google.com, www.facebook.com and www.youtube.com. Even a few graphics files have been recovered indicating that the Internet Explorer has visited these web sites.
Each file of the 22 files from the initial snapshot has been compared to its matching file from snapshot X. The evaluation has shown that at least one of the file object attribute values has changed, except the file name.
Among the 22 files there have been, e.g. counters.dat, cversions.1.db, msimgsiz.dat. The inspection of these binary files has revealed no digital evidence regarding the visited web sites.

***Chrome*** According to table 5.1, Procmon has monitored 204 distinct files which have been accessed on the virtual hard disk via Chrome's public browsing mode. Further analysis of the virtual machine disk images has shown that 63 files have been found in snapshot X and 63 files have been contained in both snapshots. Examining the disk image has revealed certain sqlite database files which Chrome either has generated or modified during the browsing session.
Digital traces regarding www.google.com, www.facebook.com and www.youtube.com have been recovered from the following sqlite database files: Current Session, Current Tabs, Favicons, Cookies, Cookies-journal, History, History Provider Cache, Network Action Predictor, Network Action Predictor-journal, Origin Bound Certs, Shortcuts and Shortcuts-journal. These database files have been among the modified files, which basically means that a particular sqlite database file from the initial snapshot has had a different file content as the corresponding file from snapshot X. Other digital evidence concerning www.youtube.com and www.google.com has been found, for example, in a file named 000041.log, that Chrome possibly uses for logging.

***Firefox*** Investigating Firefox's public browsing mode has resulted in the following findings: Procmon has registered 91 distinct files, whereof 83 have been contained in snapshot X, as presented in table 5.1. The files counters.dat and cversions.1.db have been contained in both snapshots. Both files are typically used by the Internet Explorer as well. An examination of these files has not revealed digital evidence that has been connected to any of the visited web sites.
Digital evidence pointing to www.google.com, www.facebook.com and www.youtube.com has been contained in Firefox's cache files, named _CACHE_001_, _CACHE_002_ and _CACHE_003_

under path /Users/pc/AppData/Local/Mozilla/Firefox/Profiles/4vsne8e5.default/Cache. Additionally, Firefox's sqlite databases, e.g. cookies.sqlite, permissions.sqlite and places.sqlite, have revealed information about the visited web sites.

***Safari*** The digital investigation of Safari's public browsing mode has resulted in 52 distinct files, as can be seen in table 5.1. During the browsing session no files have been modified, although Safari has created 46 files on the snapshot X disk image. The files Cache.db, Cookies.binarycookies, History.plist, LastSession.plist, WebpageIcons.db have contained digital evidence indicating that Safari has visited www.google.com, www.youtube.com and www.facebook.com.

***Opera*** When assessing Opera's public browsing artefacts, 160 different files have been found locally based on the Procmon log files, whereof 12 files have been contained in both snapshots and 134 files have been found in snapshot X, as shown in table 5.1. Opera has modified the cversions.1.db file, as well as some files that have been located under path /Users/pc/AppData/LocalLow/Microsoft/CryptnetUrlCache/Content, which have had no well known file extension (similar to those files contained under the directory the Internet Explorer /CryptnetUrlCache). When examining Opera's Internet artefacts, several graphics files have been found, such as favicons of www.youtube.com and www.facebook.com. Additionally, digital evidence regarding the three web sites has been contained in the sqlite database files Cookies, data_1, data_3, Favicons, Favicons-journal, History and History Provider Cache.

### Private Browsing Artefacts

The following section contains the analysis result of ten test runs regarding the private browsing mode of the Internet Explorer, Chrome, Firefox, Safari and Opera. Table 5.2 shows the test results of those five test runs, which have been conducted by visiting Alexa's top three web sites, whereas table 5.3 contains the results of those five test runs, that have been performed by visiting Alexa's top 300 web sites. As seen in both tables, the test results vary depending on the visited web sites as well as on the tested web browsers.

| Web Browser | Files With Inodes | Snapshot X | Snapshot X / Initial Snapshot |
|---|---|---|---|
| Internet Explorer | 135 | 92 | 30 |
| Chrome | 116 | 17 | 33 |
| Firefox | 57 | 51 | 2 |
| Safari | 80 | 73 | 0 |
| Opera | 79 | 53 | 12 |

**Table 5.2:** Private browsing artefacts generated by the Internet Explorer, Chrome, Firefox, Safari and Opera when tested Alexa's top three web sites manually

| **Web Browser** | **Files With Inodes** | **Snapshot X** | **Snapshot X / Initial Snapshot** |
|---|---|---|---|
| Internet Explorer | 18123 | 17006 | 1048 |
| Chrome | 584 | 135 | 144 |
| Firefox | 83 | 76 | 2 |
| Safari | 87 | 75 | 5 |
| Opera | 364 | 260 | 88 |

**Table 5.3:** Private browsing artefacts generated by the Internet Explorer, Chrome, Firefox, Safari and Opera when tested Alexa's top 300 web sites manually

***Internet Explorer*** Generally when analysing the private browsing mode of the Internet Explorer, a different behaviour to its public mode has been observed. As can be seen in table 5.2, the Procmon log files have contained 135 distinct files, whereof 92 files have been found in snapshot X and 30 files have been contained in both snapshots. As compared to the Internet Explorer's public mode, its private mode has not stored cookies to the snapshot X disk image. Among the files from snapshot X a file, called {93E34706-CCC5-11E3-8B9B-080027776A60}.dat, has been detected. Table 5.4 describes the attributes of this file. It has revealed traces of

| **Internet Artefact** | **Attribute** | **Value** |
|---|---|---|
| {93E34706-CCC5-11E3-8B9B-080027776A60}.dat | Path | /Users/pc/AppData/Local/Microsoft/ Internet Explorer/Recovery/High/Active |
| | MAC | mtime: 2014-04-25 22:06:09 |
| | | atime: 2014-04-25 22:06:09 |
| | | ctime: 2014-04-25 22:06:09 |
| | | crtime: 2014-04-25 22:06:09 |
| | SHA1 | b8a40d991c795355479f868365f17f0ae6c85ec4 |

**Table 5.4:** Private browsing artefact generated by the Internet Explorer pointing to www.google.com, www.facebook.com and www.youtube.com

www.google.com, www.facebook.com and www.youtube.com. In addition to this finding, several files have been recovered indicating that the Internet Explorer has called these three URLs. Table 5.5 shows those artefacts which have pointed to www.youtube.com.

| Internet Artefact | Attribute | Value |
|---|---|---|
| 1[1].png | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/FX53N95V |
| | MAC | mtime: 2014-04-25 22:05:57 |
| | | atime: 2014-04-25 22:05:57 |
| | | ctime: 2014-04-25 22:05:57 |
| | | crtime: 2014-04-25 22:05:57 |
| | SHA1 | 1673954e58303966f6f3475a63d2c0840710e3d6 |
| 1[2].png | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/FX53N95V |
| | MAC | mtime: 2014-04-25 22:05:57 |
| | | atime: 2014-04-25 22:05:57 |
| | | ctime: 2014-04-25 22:05:57 |
| | | crtime: 2014-04-25 22:05:57 |
| | SHA1 | 17fe17b7bdb5080c4bd09de9a29ca86023cae5e6 |
| 1[4].png | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/FX53N95V |
| | MAC | mtime: 2014-04-25 22:05:57 |
| | | atime: 2014-04-25 22:05:57 |
| | | ctime: 2014-04-25 22:05:57 |
| | | crtime: 2014-04-25 22:05:57 |
| | SHA1 | ee59ce5f9a49080aac6906b1be5bb150aa8436fd |
| 1NYPWD5C.htm | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/2KHPLC4H |
| | MAC | mtime: 2014-04-25 22:05:54 |
| | | atime: 2014-04-25 22:05:54 |
| | | ctime: 2014-04-25 22:05:54 |
| | | crtime: 2014-04-25 22:05:54 |
| | SHA1 | 698fe43dceac73a4bd2b9d84ee0028ca628ec816 |
| player-common-vflYYQ18S[1].png | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/6RQ61YAG |
| | MAC | mtime: 2014-04-25 22:05:59 |
| | | atime: 2014-04-25 22:05:59 |
| | | ctime: 2014-04-25 22:05:59 |
| | | crtime: 2014-04-25 22:05:59 |
| | SHA1 | 56a52efdbe1b45d1b9b927eb6432e093b01090c8 |

**Table 5.5:** Private browsing artefacts generated by the Internet Explorer referring to www.youtube.com

Table 5.6 presents those Internet artefacts that have referred to www.google.com. As can be seen, this table contains the artefact logo11w[1].png, which has been found in the snapshot X disk image as well as in the initial snapshot disk image. This table entry shows that the file has been modified during the private browsing session, as indicated by the modified MAC times and SHA1 sums. Table 5.7 shows the Internet artefact which has pointed to www.facebook.com.

| **Internet Artefact** | **Attribute** | **Value** |
|---|---|---|
| favicon[1].ico | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/H1UEFVG3 |
| | MAC | mtime: 2014-04-09 18:35:20 |
| | | atime: 2014-04-09 18:35:20 |
| | | ctime: 2014-04-09 18:35:20 |
| | | crtime: 2014-04-09 18:35:20 |
| | SHA1 | e2020bf4f2b65f62434c62ea967973140b3300df |
| nav_logo193[1].png | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/9M0V594T |
| | MAC | mtime: 2014-04-25 22:05:42 |
| | | atime: 2014-04-25 22:05:42 |
| | | ctime: 2014-04-25 22:05:42 |
| | | crtime: 2014-04-25 22:05:42 |
| | SHA1 | fb409adb53e544dbf6938bd13e02d51d2ab621a0 |
| logo11w[1].png | Path | Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/E21RIXID |
| | MAC | X mtime: 2014-04-25 22:06:56<br>Init mtime: 2014-03-28 23:24:51 |
| | | X atime: 2014-04-25 22:06:56<br>Init atime: 2014-03-28 23:24:51 |
| | | X ctime: 2014-04-25 22:06:56<br>Init ctime: 2014-03-28 23:24:51 |
| | | X crtime: 2014-04-25 22:06:56<br>Init crtime: 2014-03-28 23:24:51 |
| | SHA1 | X: 68cb9af63939e1d509d34a5bf92fb2824acb64c3<br>Init: 349841408d1aa1f5a8892686fbdf54777afc0b2c |

**Table 5.6:** Private browsing artefacts generated by the Internet Explorer pointing to www.google.com

| Internet Artefact | Attribute | Value |
|---|---|---|
| pNXXLTrgcZK[1].png | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/P8LC0BGK |
| | MAC | mtime: 2014-04-25 22:05:49 |
| | | atime: 2014-04-25 22:05:49 |
| | | ctime: 2014-04-25 22:05:49 |
| | | crtime: 2014-04-25 22:05:49 |
| | SHA1 | 41eb03fcd7db597b9ea904fb0d05eeaa5f709d11 |

**Table 5.7:** Private browsing artefact generated by the Internet Explorer pointing to www.facebook.com

When evaluating the test results regarding Alexa's top 300 web sites, the number of files with inodes in table 5.3 is different as compared to those from table 5.2. As seen in table 5.3, the Procmon log files have contained 18123 distinct files, whereof 17006 files have been found in snapshot X and 1048 files have been detected in snapshot X as well as in the initial snapshot. The Internet Explorer's private mode has not created any cookie files, which is similar to the behaviour that has been observed when analysing the test result regarding Alexa's top three web sites.
Similar to the previous evaluation of the Internet Explorer's private browsing mode, a file called {A22C12BB-E713-11E3-BA39-080027776A60}.dat, has been recovered from the snapshot X disk image, as presented in table 5.8.

| Internet Artefact | Attribute | Value |
|---|---|---|
| {A22C12BB-E713-11E3-BA39-080027776A60}.dat | Path | /Users/pc/AppData/Local/Microsoft/Internet Explorer/ Recovery/High/Active |
| | MAC | mtime: 2014-05-29 09:58:06 |
| | | atime: 2014-05-29 09:58:06 |
| | | ctime: 2014-05-29 09:58:06 |
| | | crtime: 2014-05-29 09:58:06 |
| | SHA1 | f6e37172a6c788c56defdd1f8daadca435995904 |

**Table 5.8:** Private browsing artefact generated by the Internet Explorer containing digital evidence regarding Alexa's top 300 web sites when tested manually

This file has exposed digital evidence concerning the following URLs, as seen in table 5.9.

| | | |
|---|---|---|
| www.doublepimp.com | www.milliyet.com.tr | www.flickr.com |
| www.photobucket.com | www.twitter.com | www.alibaba.com |
| www.godaddy.com | www.bing.com | www.odnoklassniki.ru |
| www.ndtv.com | www.baidu.com | www.rambler.ru |
| www.pandora.com | www.goo.ne.jp | www.chinaz.com |
| www.lenta.ru | www.facebook.com | www.mail.ru |
| www.rbc.ru | www.popads.net | www.google.be |
| www.bleacherreport.com | www.twitch.tv | www.ettoday.net |
| www.wp.pl | www.google.ro | www.moz.com |
| www.avito.ru | www.businessinsider.com | www.gmx.net |
| www.meetup.com | www.fedex.com | www.39.net |
| www.samsung.com | www.douban.com | www.stackexchange.com |
| www.4dsply.com | www time.com | www.mercadolivre.com.br |
| www.disqus.com | www.mama.cn | www.google.pt |
| www.usatoday.com | www.youtube.com | www.blogspot.com |
| www.uploaded.net | www.constantcontact.com | www.bitly.com |
| www.avito.ru | www.leboncoin.fr | www.ucoz.ru |
| www.youyuan.com | www.mashable.com | www.washingtonpost.com |

**Table 5.9:** URLs revealed by the Internet Explorer's private browsing mode when tested Alexa's top 300 web sites manually

During the digital investigation various file types have been detected on the snapshot X disk image. A list of the file types that the Internet Explorer has accessed most frequently in the course of this specific test run is presented in table 5.10. According to each file type, a different number of items (files) has been recovered. Among these files there have been many graphics files, such as JPEGs, GIFs and PNGs, but also HTML, CSS and Javascript files have been found. The Internet artefacts contained in table 5.11 and in table 5.12 have revealed digital traces by

| File Type | Items | File Type | Items | File Type | Items | File Type | Items |
|---|---|---|---|---|---|---|---|
| jpg | 6692 | css | 837 | svg | 57 | cab | 8 |
| png | 3366 | txt | 192 | tmp | 23 | cms | 6 |
| js | 2809 | eot | 171 | ico | 13 | swf | 4 |
| gif | 2289 | woff | 107 | dat | 12 | db | 2 |
| htm | 917 | json | 97 | xml | 10 | *others* | 442 |

**Table 5.10:** File types created by the Internet Explorer's private browsing mode when tested Alexa's top 300 web sites manually

referring to several URLs of the tested web sites.

| Artefact | Attribute | Value |
|---|---|---|
| integ1[1].jpg | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/K7AVE189 |
| | MAC | mtime: 2014-05-29 09:56:33 |
| | | atime: 2014-05-29 09:56:33 |
| | | ctime: 2014-05-29 09:56:33 |
| | | crtime: 2014-05-29 09:56:33 |
| | SHA1 | 13cad9ed2c9d4bd1738566e32e30f004885bcb34 |
| twttr[1].png | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/H1UEFVG3 |
| | MAC | mtime: 2014-05-29 09:47:42 |
| | | atime: 2014-05-29 09:47:42 |
| | | ctime: 2014-05-29 09:47:42 |
| | | crtime: 2014-05-29 09:47:42 |
| | SHA1 | c58494bd373cdd3f99ce61be77f52ec647d40ef0 |
| unlogo03[1].gif | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/UFKPZ290 |
| | MAC | mtime: 2014-05-29 09:48:58 |
| | | atime: 2014-05-29 09:48:58 |
| | | ctime: 2014-05-29 09:48:58 |
| | | crtime: 2014-05-29 09:48:58 |
| | SHA1 | 1903290203634f92528c1d316bfaa373354cd152 |
| yahoo-min[1].js | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/H1UEFVG3 |
| | MAC | mtime: 2014-05-29 09:51:46 |
| | | atime: 2014-05-29 09:51:46 |
| | | ctime: 2014-05-29 09:51:46 |
| | | crtime: 2014-05-29 09:51:46 |
| | SHA1 | 9ec06e59640219090d69e016ed21bf7f47b75836 |
| VAOL0OJO.htm | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/8UOXYNFW |
| | MAC | mtime: 2014-05-29 09:56:12 |
| | | atime: 2014-05-29 09:56:12 |
| | | ctime: 2014-05-29 09:56:12 |
| | | crtime: 2014-05-29 09:56:12 |
| | SHA1 | 4e00340a692954aa5d6ec0f1602bafe6ed2485a3 |

**Table 5.11:** Private browsing artefacts generated by the Internet Explorer referring to www.paypal.com, www.twitter.com, www.baidu.com, www.yahoo.com and www.washingtonpost.com when tested Alexa's top 300 web sites manually

| Artefact | Attribute | Value |
|---|---|---|
| f[1].txt | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/8UOXYNFW |
| | MAC | mtime: 2014-05-29 09:56:14 |
| | | atime: 2014-05-29 09:56:14 |
| | | ctime: 2014-05-29 09:56:14 |
| | | crtime: 2014-05-29 09:56:14 |
| | SHA1 | 0ec5c4642ca4528c395b011093b86a0536658252 |
| p[1].json | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/IMPBATQ6 |
| | MAC | mtime: 2014-05-29 09:42:57 |
| | | atime: 2014-05-29 09:42:57 |
| | | ctime: 2014-05-29 09:42:57 |
| | | crtime: 2014-05-29 09:42:57 |
| | SHA1 | e0c5339165ec60e447b059a62fec3351935c5921 |
| ico_linkedin[1].svg | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/8EF7Q49B |
| | MAC | mtime: 2014-05-29 09:35:26 |
| | | atime: 2014-05-29 09:35:26 |
| | | ctime: 2014-05-29 09:35:26 |
| | | crtime: 2014-05-29 09:35:26 |
| | SHA1 | 45e96b1238f2b5fd6d0e882c01d8b8bee87266a3 |
| dat9713.tmp | Path | /Users/pc/AppData/Local/Temp |
| | MAC | mtime: 2014-05-29 09:56:55 |
| | | atime: 2014-05-29 09:56:55 |
| | | ctime: 2014-05-29 09:56:55 |
| | | crtime: 2014-05-29 09:56:55 |
| | SHA1 | 3573ce02b7e4ce86e6445048d7291673c84d89c2 |
| sojus-dockt-an-iss-an-36181590,view= videoChannelXml.bild[1] .xml | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/X174NYMH |
| | MAC | mtime: 2014-05-29 09:51:19 |
| | | atime: 2014-05-29 09:51:19 |
| | | ctime: 2014-05-29 09:51:19 |
| | | crtime: 2014-05-29 09:51:19 |
| | SHA1 | 7a98ad660e031bb12e443a89c145133b9a36e7bb |

**Table 5.12:** Private browsing artefacts generated by the Internet Explorer referring to www.washingtonpost.com, www.indiatimes.com, www.linkedin.com, www.wordpress.com and www.bild.de when tested Alexa's top 300 web sites manually

These URLs and their related Internet artefacts (file types) are connected as follows:

- integ1[1].jpg → www.paypal.com
- twttr[1].png → www.twitter.com
- unlogo03[1].gif → www.baidu.com
- yahoo-min[1].js → www.yahoo.com
- VAOL0OJO.htm → www.washingtonpost.com
- f[1].txt → www.washingtonpost.com
- p[1].json → www.indiatimes.com
- ico_linkedin[1].svg → www.linkedin.com
- dat9713.tmp → www.wordpress.com
- sojus-dockt-an-iss-an-36181590,view=videoChannelXml.bild[1].xml → www.bild.de

For example, when the Internet Explorer has visited www.washingtonpost.com in private mode, several Internet artefacts have been accessed on the local hard disk, as indicated by the findings in table 5.8, in table 5.11 and in table 5.12. Obviously, this means that investigators will be able to recover more Internet artefacts referring to the same web site in the course of a forensic examination.
No digital evidence has been detected by analysing files of the following types: eot, woff, cab, css, cms, swf, db and ico. The term *others* in table 5.10 includes files with no well known file suffix like the file
0B1B84E1509125064E3D44331C3817C2_AF3AB4EE7CFAD73E2B899C1EB105F1B2. Typically, these files are in binary format. No digital evidence regarding the tested web sites has been found by examining such files. As can be seen, the evaluation of this test run has yielded in similar results as compared to those results when tested Alexa's top three web sites.

***Chrome*** When investigating Chrome's private browsing mode, 116 distinct files have been listed in the Procmon log files, whereof 17 files have been found in snapshot X and 33 have been contained in both snapshots. This is shown in table 5.2. In contrast to Chrome's public mode, the private mode has not revealed user-related browsing data. Although sqlite database files, such as Cookies, Cookies-journal, History Provider Cache and Network Action Predictor, have had modified file attributes (e.g. MAC times, hash digests), the examination of these files has yielded in no valuable information about the visited web sites.
The evaluation of Chrome's test result concerning Alexa's top 300 web sites has yielded in the following findings: 584 distinct files have been listed in the Procmon log files, whereof 135 files have been found in snapshot X and 144 have been detected in both snapshots. This is shown in table 5.3. As a result, the examination has not revealed any user-related browsing data. Although Chrome's sqlite database files, such as History, History-journal, History Provider

Cache, Shortcuts and Web Data, have had modified file attributes, the investigation of those files has not resulted in any digital evidence about the tested web sites. The evaluation of this test run indicates that Chrome behaves similarly as compared to the test results regarding Alexa's top three web sites.

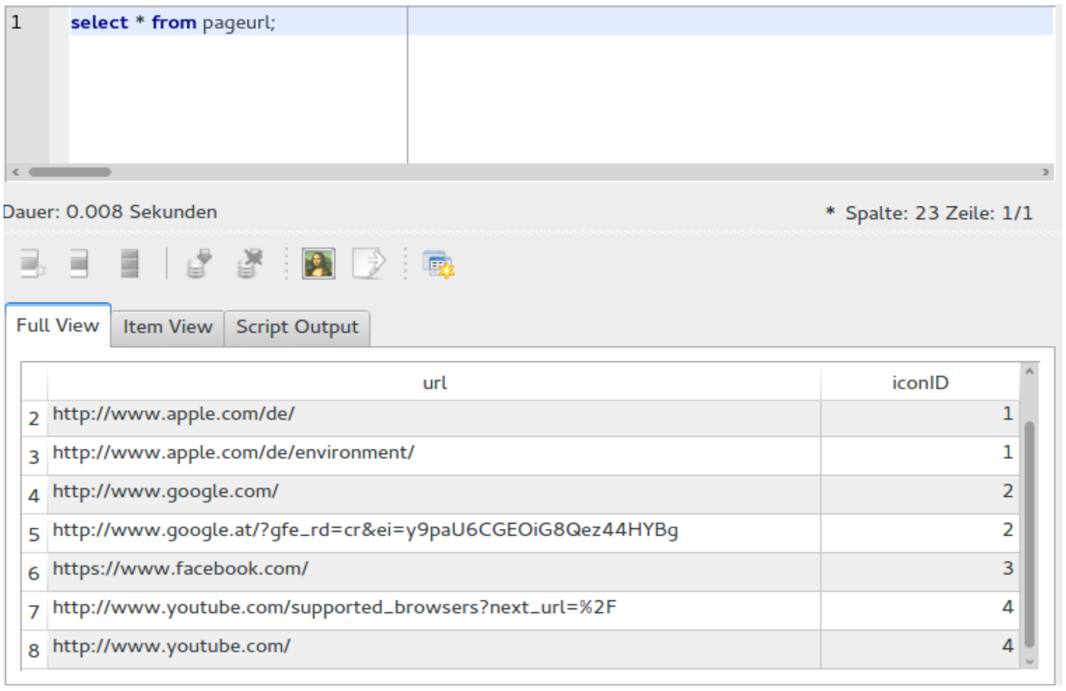
***Firefox*** According to table 5.2, Procmon has reported 57 distinct files which Firefox has accessed locally in private browsing mode. 51 files have been recovered from the snapshot X disk image. The file contents of counters.dat and cversions.1.db have been modified, which is similar to the behaviour that has been observed when investigating Firefox's public mode. In contrast to Firefox's public browsing mode, its private mode has not revealed any valuable information about the visited web sites. The files _CACHE_001_, _CACHE_002_ and _CACHE_003_ under path /Users/pc/AppData/Local/Mozilla/Firefox/Profiles/4vsne8e5.default/Cache have not contained any user-related browsing data. Another difference between the public and the private mode is that when using the latter one, Firefox purges graphics files on web browser shutdown. Additionally, the private mode cleans that user-related browsing content which is contained in the sqlite databases.
When analysing the private browsing artefacts according to Alexa's top 300 web sites, there has been no big difference, as compared to the test results regarding Alexa's top three web sites, except the number of files which have been created locally. As seen in table 5.3, Procmon has reported 83 distinct file names which Firefox has accessed in private browsing mode. 76 files have been recovered from snapshot X and two files have been salvaged from both snapshots. The evaluation of the recovered files has not exposed any user-related browsing data.
The investigation of both test runs has yielded in no valuable information about the tested web sites.

***Safari*** In accordance with table 5.2, the examination of Safari's private mode has revealed 80 distinct files, whereof 73 files have been found in snapshot X. Contrary to the evaluation of Safari's public browsing mode, the files Cache.db, Cookies.binarycookies, History.plist and LastSession.plist have not contained any digital evidence pointing to the visited web sites. Further analysis has revealed user-related browsing data which has been contained in the file WebpageIcons.db. Table 5.13 shows this file, its attributes and values. The file has contained URLs referring to www.google.com, www.facebook.com and www.youtube.com. Additionally, figure

| Artefact | Attribute | Value |
|---|---|---|
| WebpageIcons.db | Path | /Users/pc/AppData/Local/Apple Computer/Safari |
| | MAC | mtime: 2014-04-25 22:00:04 |
| | | atime: 2014-04-25 22:00:04 |
| | | ctime: 2014-04-25 22:00:04 |
| | | crtime: 2014-04-25 22:00:04 |
| | SHA1 | 47f57867eba5fde59376029becce0f8cd47240fd |

**Table 5.13:** Private browsing artefact generated by Safari containing digital evidence regarding www.google.com, www.facebook.com and www.youtube.com

5.1 illustrates the content of the sqlite database table pageurl, that is part of the WebpageIcons.db file. As can be seen, digital traces are contained in this table which point to Alexa's top three



```
select * from pageurl;
```

Dauer: 0.008 Sekunden * Spalte: 23 Zeile: 1/1

| | url | iconID |
|---|---|---|
| 2 | http://www.apple.com/de/ | 1 |
| 3 | http://www.apple.com/de/environment/ | 1 |
| 4 | http://www.google.com/ | 2 |
| 5 | http://www.google.at/?gfe_rd=cr&ei=y9paU6CGEOiG8Qez44HYBg | 2 |
| 6 | https://www.facebook.com/ | 3 |
| 7 | http://www.youtube.com/supported_browsers?next_url=%2F | 4 |
| 8 | http://www.youtube.com/ | 4 |

**Figure 5.1:** Sample view of Safari's sqlite database table pageurl revealing digital evidence concerning www.google.com, www.facebook.com and www.youtube.com

web sites.
Contrary to the test result concerning Alexa's top three web sites, the examination of Safari's private mode regarding Alexa's top 300 web sites has revealed 87 distinct file names, whereof 75 files have been contained in snapshot X. Five files have been found in both snapshots. Table 5.3 shows this test result. Again the evaluation of this test run has exposed user-related browsing data which has been located in the file WebpageIcons.db. Table 5.14 shows the file and its attributes. Similar to the Internet Explorer file {A22C12BB-E713-11E3-BA39-080027776A60}.dat from before, the WebpageIcons.db file has revealed digital traces regarding the web sites, as shown in table 5.15. No digital evidence has been found by inspecting the remaining browsing artefacts.

***Opera*** Prcomon's log files have contained 79 distinct files regarding Opera's private browsing session, as seen in table 5.2. 53 files have been found in snapshot X, whereas 12 files have been contained in both snapshots. Similar to Opera's public mode, its private mode has modified the cversions.1.db file and some other files under path /Users/pc/AppData/LocalLow/Microsoft/CryptnetUrlCache/Content. No valuable digital evidence has been exposed by the private browsing mode, although it has accessed several browsing artefacts on the local hard disk, e.g. Archived History, Visited Links, Web Data and stash.db.
The evaluation of Alexa's top 300 web sites has yielded in similar findings: Procmon's log files

| Artefact | Attribute | Value |
|---|---|---|
| WebpageIcons.db | Path | /Users/pc/AppData/Local/Apple Computer/Safari |
| | MAC | mtime: 2014-05-29 14:34:21 |
| | | atime: 2014-05-29 14:34:21 |
| | | ctime: 2014-05-29 14:34:21 |
| | | crtime: 2014-05-29 14:34:21 |
| | SHA1 | c8161c625ec252b2de883c9236881c41e49cbb4d |

**Table 5.14:** Private browsing artefact generated by Safari containing digital evidence regarding regarding Alexa's top 300 web sites when tested manually

| | | |
|---|---|---|
| www.4dsply.com | www.amazon.fr | www.apple.com |
| www.ask.com | www.avito.ru | www.baidu.com |
| www.bing.com | www.blogspot.in | www.domaintools.com |
| www.ebay.de | www.facebook.com | www.fedex.com |
| www.goodgamestudios.com | www.google.at | www.google.be |
| www.google.ca | www.google.ch | www.google.cl |
| www.google.co.in | www.google.co.jp | www.google.com |
| www.google.com.bd | www.google.com.hk | www.google.com.ng |
| www.google.com.ph | www.google.com.sg | www.google.com.tr |
| www.google.com.tw | www.google.co.th | www.google.co.uk |
| www.google.co.ve | www.google.co.za | www.google.dz |
| www.google.fr | www.google.gr | www.google.it |
| www.google.pt | www.google.ro | www.google.ru |
| www.ikea.com | www.leboncoin.fr | www.mail.ru |
| www.meetup.com | www.msn.com | www.myntra.com |
| www.onclickads.net | www.rediff.com | www.snapdeal.com |
| www.t.co | www.thefreecamsecret.com | www.tripadvisor.com |
| www.ups.com | www.vube.com | www.w3schools.com |
| www.warriorforum.com | www.wikihow.com | www.wikipedia.org |
| www.wordpress.org | www.wordreference.com | www.yandex.ru |

**Table 5.15:** URLs revealed by Safari's private browsing mode when tested Alexa's top 300 web sites manually

have contained 364 distinct file names regarding Opera's private browsing session, as shown in table 5.3. 260 files have been contained in snapshot X, whereas 88 files have been found in both snapshots. Although Opera's sqlite databases, such as Favicons, History, History-journal, History Provider Cache, thumbnails.db or Web Data, and its cache files, such as data_0, data_1 or data_2, have been accessed during the test run, the private mode has not revealed user-related browsing information.

### 5.4.3 Evaluation of the Automated Test Scenarios

This section presents the evaluation of the automated test results regarding the Internet artefacts which have been generated by the private browsing mode of the Internet Explorer and Chrome.

#### Private Browsing Artefacts

As can be seen in table 5.16 the testing results of the Internet Explorer and Chrome are similar to those in the table 5.3. Approximately 21500 Internet artefacts have been detected when evalu-

| Web Browser | Files With Inodes | Snapshot X | Snapshot X / Initial Snapshot |
|---|---|---|---|
| Internet Explorer | 18984 | 17282 | 1312 |
| Internet Explorer | 23045 | 22198 | 512 |
| Internet Explorer | 22632 | 21734 | 581 |
| Internet Explorer | 22457 | 21639 | 570 |
| Chrome | 394 | 74 | 182 |
| Chrome | 171 | 57 | 14 |
| Chrome | 261 | 87 | 91 |
| Chrome | 263 | 79 | 116 |

**Table 5.16:** Private browsing artefacts generated by the Internet Explorer and Chrome when tested Alexa's top 300 web sites automatically

ating the four test runs of the Internet Explorer, which is a quite similar behaviour as it has been observed when evaluating the manual test results. In contrast to the Internet Explorer, Chrome has not revealed user related browsing data, although browsing artefacts have been accessed, such as the sqlite database files History, Network Predictor, Visited Links or Web Data. At an average, 175 of Chrome's private browsing artefacts have been recovered.
As the evaluation of both web browsers has yielded in findings that have been similar to the findings of the manual testing phase, the execution of further test runs has been omitted. A detailed presentation of test run two to four of the Internet Explorer and those test runs of Chrome has been neglected, since further analysis has yielded in a similar outcome as compared to the outcome from section 5.4.2. Therefore, the following section only presents the first test run of the Internet Explorer in more detail. This test run has been chosen to point out the similarities / differences between the manual and the automated execution of the Internet Explorer's private browsing mode.

***Internet Explorer*** As can be seen in table 5.17, the Internet Explorer has created items (files) of a similar file type as compared to those items from table 5.10.

| File Type | Items | File Type | Items | File Type | Items | File Type | Items |
|---|---|---|---|---|---|---|---|
| jpg | 6771 | htm | 860 | svg | 40 | swf | 5 |
| png | 3598 | eot | 140 | txt | 21 | cab | 4 |
| js | 3436 | woff | 106 | dat | 8 | cms | 3 |
| gif | 2459 | json | 102 | xml | 7 | db | 2 |
| css | 879 | tmp | 42 | ico | 7 | *ohters* | 104 |

**Table 5.17:** File types created by the Internet Explorer's private browsing mode when tested Alexa's top 300 web sites automatically

The Internet artefacts presented in table 5.18 and in table 5.19 have revealed digital traces regarding the following web sites:

- integ8[1].jpg → www.paypal.com
- twttr[1].png → www.twitter.com
- unlogo03[1].gif → www.baidu.com
- yahoo-min[1].js → www.yahoo.com
- LJ7WZAV5.htm → www.washingtonpost.com
- yerelhaberler[1].txt → www.milliyet.com
- q[1].json → www.rakuten.co.jp
- google_follow[1].svg → www.google.com
- dat26F8.tmp → www.twitter.com
- -30373542,sort=1,n=1,view=rss2.bild[1].xml → www.bild.de

As can be seen, the automated execution of the Internet Explorer's private mode has resulted in artefacts that are identical to those which have been detected when testing Alexa's top 300 web sites manually. For example, the files twttr[1].png, unlogo03[1].gif and yahoo-min[1].js have referred to the same web sites as their equivalents from section 5.4.2. Furthermore, these artefacts and their counterparts have had the same SHA1 sums, as presented in table 5.11 and table 5.18. As has been observed before, the Internet Explorer's private mode will create several browsing artefacts which are related to a single web sites. For instance, the files twttr[1].png and dat26F8.tmp have referred to www.twitter.com.
No digital evidence has been detected by examining the following file types: css, eot, woff, dat, ico, swf, cab, cms and db. The file type *others* can be understood in a similar manner as described in section 5.4.2.
A difference between the manual and automated execution of test runs is that the automated execution of Internet Explorer's private browsing mode has not left digital traces in .dat files. This behaviour has been observed when analysing the remaining three automated test runs of the Internet Explorer as well.

| Artefact | Attribute | Value |
|---|---|---|
| integ8[1].jpg | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/2LDTUL6S |
| | MAC | mtime: 2014-03-29 06:48:59 |
| | | atime: 2014-03-29 06:48:59 |
| | | ctime:2014-03-29 06:48:59 |
| | | crtime: 2014-03-29 06:48:59 |
| | SHA1 | c8d7f6b9846785eda50040c9b5f971543b2bd6ef |
| twttr[1].png | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/NI9JJ69Q |
| | MAC | mtime: 2014-03-29 06:37:37 |
| | | atime: 2014-03-29 06:37:37 |
| | | ctime: 2014-03-29 06:37:37 |
| | | crtime: 2014-03-29 06:37:37 |
| | SHA1 | c58494bd373cdd3f99ce61be77f52ec647d40ef0 |
| unlogo03[1].gif | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/9M0V594T |
| | MAC | mtime: 2014-03-29 06:38:54 |
| | | atime: 2014-03-29 06:38:54 |
| | | ctime: 2014-03-29 06:38:54 |
| | | crtime: 2014-03-29 06:38:54 |
| | SHA1 | 1903290203634f92528c1d316bfaa373354cd152 |
| yahoo-min[1].js | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/9M0V594T |
| | MAC | mtime: 2014-03-29 06:41:25 |
| | | atime: 2014-03-29 06:41:25 |
| | | ctime: 2014-03-29 06:41:25 |
| | | crtime: 2014-03-29 06:41:25 |
| | SHA1 | 9ec06e59640219090d69e016ed21bf7f47b75836 |
| LJ7WZAV5.htm | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/YVRY29JX |
| | MAC | mtime: 2014-03-29 06:48:30 |
| | | atime: 2014-03-29 06:48:30 |
| | | ctime: 2014-03-29 06:48:30 |
| | | crtime: 2014-03-29 06:48:30 |
| | SHA1 | 448194b8c17a9aa645c87946e54d7f08d78ace3c |

**Table 5.18:** Private browsing artefacts generated by the Internet Explorer referring to www.paypal.com, www.twitter.com, www.baidu.com, www.yahoo.com and www.washingtonpost.com when tested Alexa's top 300 web sites automatically

| Artefact | Attribute | Value |
|---|---|---|
| yerelhaberler[1].txt | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/YVRY29JX |
| | MAC | mtime: 2014-03-29 06:48:22 |
| | | atime: 2014-03-29 06:48:22 |
| | | ctime: 2014-03-29 06:48:22 |
| | | crtime: 2014-03-29 06:48:22 |
| | SHA1 | 3930a73e6fade04c1bc93845ea9c006c44178e73 |
| q[1].json | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/RWDCNZFL |
| | MAC | mtime: 2014-03-29 06:32:03 |
| | | atime: 2014-03-29 06:32:03 |
| | | ctime: 2014-03-29 06:32:03 |
| | | crtime: 2014-03-29 06:32:03 |
| | SHA1 | e98eb5bb6760e62838c37c6b7ed38b547f2d0c2e |
| google_follow[1].svg | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/P8LC0BGK |
| | MAC | mtime: 2014-03-29 06:38:08 |
| | | atime: 2014-03-29 06:38:08 |
| | | ctime: 2014-03-29 06:38:08 |
| | | crtime: 2014-03-29 06:38:08 |
| | SHA1 | c552fd5b5bd2cc3937c38cb41a913415dad29b2f |
| dat26F8.tmp | Path | /Users/pc/AppData/Local/Temp |
| | MAC | mtime: 2014-03-29 06:49:29 |
| | | atime: 2014-03-29 06:49:29 |
| | | ctime: 2014-03-29 06:49:29 |
| | | crtime: 2014-03-29 06:49:29 |
| | SHA1 | f39a8d4aa32bc495d77f47a0df42d2d2f11aa07d |
| -30373542,sort=1, n=1,view=rss2.bild[1] .xml | Path | /Users/pc/AppData/Local/Microsoft/Windows/ Temporary Internet Files/Content.IE5/IMPBATQ6 |
| | MAC | mtime: 2014-03-29 06:40:49 |
| | | atime: 2014-03-29 06:40:49 |
| | | ctime: 2014-03-29 06:40:49 |
| | | crtime: 2014-03-29 06:40:49 |
| | SHA1 | d126c744e78e14190aba20edf0f1d113cd3389de |

**Table 5.19:** Private browsing artefacts generated by the Internet Explorer referring to www.milliyet.com, www.rakuten.co.jp, www.google.com, www.twitter.com and www.bild.de when tested Alexa's top 300 web sites automatically

## 5.5 Summary

Generally, all five web browser have left digital traces on the virtual machine disk images when they had been used in private mode. Each tested web browser has written data on start-up, during a browsing session and on shutdown.
The main finding during the digital investigation has been the privacy leak in the Internet Explorer's private browsing mode. The Internet Explorer created a big amount of data on the local hard drive which has not been removed on shutdown. Therefore, many files have been recoverable. Due to the large bulk of data, a sample file per file type has been drawn for the most frequently occurred file types, in order to show that the recovered Internet artefacts are connected with the tested web sites. The evaluation of the manually tested web sites has not resulted in many differences as compared to automatically tested ones.
Basically this mean that forensic examiners will have to look for files of the following file types when investigating the private mode of modern web browsers: jpg, png, gif, js, htm, txt, json, svg, tmp, dat and xml. There may be a chance that these files bear digital evidence. As a consequence, forensic investigators can draw conclusions about the actions a user has performed on the Internet.
Another finding has been the WebpageIcons.db file when investigating Safari's private browsing mode. This sqlite database file has contained digital traces which have referred to the tested URLs. Examiners are advised to keep an eye on this file in the course of a digital investigation, as they may find digital evidence there about the web sites a user has visited.
The other web browsers have implemented their private browsing mode in a sufficient way. No relevant user-related browsing data has been exposed.

CHAPTER 6

# Discussion

Within this chapter the results of this work are discussed. The strength and the limitations of the proposed digital investigation approach are described as well as the remaining challenges and future work.

## 6.1 Discussion of the Test Results

The evaluation of the test results has been carried out according the defined test scenarios from section 5.1. In general, the evaluation of the collected data has been done by checking random samples, because the Internet Explorer has generated a big amount of test results. An in-depth inspection of all Internet Explorer browsing artefacts would have exceeded the scope of this work, which was therefore omitted. Chrome, Firefox, Safari and Opera have created a rather small amount of test results, as browsing data typically is stored to their sqlite databases.
In the first step, the public browsing artefacts of the manual test scenario have been examined. As part of this scenario, Alexa's top three web sites have been used for testing. All five web browsers have left digital evidence on the virtual machine disk images. The Internet Explorer, for example, has created cookies and various graphics files (e.g. PNGs, JPEGs) - Javascript and XML file have been detected as well. Additionally, the Internet Explorer has modified a few .dat files, where digital evidence has been found regarding the tested web sites.
Contrary to the Internet Explorer, the other web browsers have left digital traces in their specific sqlite database files. For instance, Chrome has created digital evidence by writing data to its History database, whereas Firefox has used its cache files (e.g. _CACHE_001_) to save browsing data. Similar to Firefox and Chrome, Safari and Opera have stored the browsing content to their database files, such as Cache.db (Safari) or Favicons (Opera), thereby revealing user actions on the Internet. As the analysis has shown, all web browser except the Internet Explorer are mainly using sqlite databases for caching web data. Safari, for example, also uses a .plist files which are in binary format, whereas Chrome, for instance, uses log files that are in text format.
In the second step, the private browsing artefacts of all five web browsers have been analysed.

Within this step the web browsers have been tested manually by using Alexa's top three web sites and they have been tested by the usage of Alexa' top 300 web sites. The main difference between both test scenarios is the amount of browsing data that has been logged by Procmon, as well as the number of inodes that have been recovered from the virtual machine disk images, i.e. more test results have been obtained by performing tests across a wider range of web sites.
For example, when evaluating Internet Explorer's private mode according to the first test scenario, it has not been possible to recover any cookie files, as it has been the case when analysing Internet Explorer's public mode. Nevertheless, the Internet Explorer has left a .dat file and some other files which have contained traces of Alexa's top three URLs. The remaining four web browsers have wiped out their digital traces, except Safari which has left digital evidence in the WebpageIcons.db file.
When examining Internet Explorer's private mode regarding Alexa's top 300 URLs, there has been leaked a big amount of user-related browsing data to the local hard drive. In contrast, the other web browsers - except Safari which has left digital traces in the WebpageIcons.db file as well - have not exposed sensitive user data, although they have modified and created several browsing artefacts, such as sqlite database files, log files or binary files, on the local hard disk.
In the third step, the private browsing mode of the Internet Explorer and Chrome has been tested automatically. The evaluation of the four test runs performed by the Internet Explorer has revealed many browsing artefacts. This indicates that the Internet Explorer's private mode acts similar to the observations that have been made during the manual testing phase. The evaluation of Chrome's automated test runs has exposed no valuable user-related browsing data.
As a result, it has been found that the private modes of the tested web browsers have been implemented properly, except that one which is released with the Internet Explorer. Furthermore, it has been found that there are differences from a forensically standpoint of view, such as the number of recovered files and specific file contents, when the private browsing mode is launched manually, in contrast to when it is launched automatically.
Chrome mainly has used its sqlite databases, but it has written various log files to the local hard disk as well. Firefox, Safari and Opera have not been tested yet automatically, but an analysis of the manually tested web sites has shown that their private modes act rather similar to that of Chrome.

## 6.2 Strength of This Approach

One of the key benefits of this approach is the accuracy of monitoring the actions a web browser performs on the local file system (e.g. of a virtual machine), when surfing on the Internet. Monitoring those events has been done effectively by collecting log data via Procmon.
Another advantage in contrast to the approach of classical timeline analysis is that the Procmon log files contain detailed information about those files which have been accessed during a browsing session, i.e. classical time lining compared to the examination of Procmon's log files is not effective enough, because classical timelines do not only contain information about the events a particular process has triggered on the local file system, but they comprise of all file system events (including operating system noise) that have occurred locally. For example, by using Procmon's log files a detailed history of those file system events can be obtained that a

web browser has induced on the local system. This more detailed ”timeline” can be used not only for analysing the history of particular events, but also for recovering files, which is another benefit.
Another advantage of the proposed forensic investigation approach is the detection of modified file contents by means of fiwalk and DFXML. Fiwalk is capable of generating DFXML files which can be easily compared to each other, in order to detect similarities and/or dissimilarities between specific file objects.

## 6.3 Limitations of This Approach

A limitation of this approach is the partial automation of both the testing framework as well as the forensic analysis framework. Due to the technical limitations of VirtualBox’s process control in conjunction with Procmon, testing web browsers can only be performed by manually configuring the testing framework every time before a test run. For example, a shared folder on the host and on the guest system needs to be prepared every time before a test can be launched. Additionally, the folder structure of the previous test run needs to be saved manually and the testing framework needs to be provided with new input data.
Another drawback of this approach is the limited Selenium support for Firefox, Safari and Opera, as automating these web browsers has been found to be unsuitable. Therefore, testing all five web browser in an automated way was not possible.
Another disadvantage of this approach is the big amount of browsing data, that needs to be evaluated manually, e.g. in the case of the Internet Explorer. Due to this fact, a detailed analysis of each recovered file object was not possible.

## 6.4 Remaining Challenges and Future Work

As described in the previous section, a hurdle to overcome will be the partial automation of the remaining three web browsers.
The forensic analysis framework may be extended as part of a future work by covering features like slack space analysis or file content analysis by means of file carving.
A further improvement of the overall analysis procedure will be the implementation of a database that will contain the test results of certain test runs. These test results can be analysed by using adequate data mining techniques, in order to find user-related web browsing patterns.

CHAPTER 7

# Conclusion

The goal of this work was to detect user privacy leaks in the private browsing mode of modern web browsers by applying a new digital investigation approach, which enables forensic examiners to find the digital evidence a web browser has left on the local computer system after a browsing session.
An introduction has been given to the field of digital forensics by explaining the consecutive steps of the digital forensic process, which are the acquisition, the examination and the presentation of digital evidence. As part of the acquisition phase, disk imaging techniques have been described, as well as the volatility of data, which is crucial to understand when collecting and preserving digital evidence.
The abstraction levels of file systems and slack space have been introduced, since a basic knowledge about file system features can be useful in course of a digital examination. Furthermore, common digital forensic analysis techniques have been described, which are timeline analysis, deleted file recovery, Windows Registry forensics, file carving, network forensics and physical memory forensics.
As part of the field of digital forensics, anti-forensic techniques have been presented. The goal of anti-forensics is to make digital investigations difficult, i.e. there exist many anti-forensic methods, such as data encryption or data destruction, which can be used to hamper a digital investigation. Due to the fact that private web browsing uses anti-forensic techniques as well, an introduction to this topic has been presented. Additionally, the related work section in this work discusses several approaches that have been applied to counterfeit those anti-forensic techniques which modern web browsers use for enhancing user privacy.
For the implementation of the proposed forensic analysis approach, the theoretical concepts of timeline analysis and delete file recovery have been used. The remaining investigation techniques mentioned in this work may be part of a future work, as implementing them would have exceeded the scope of this work.
The practical part of this work takes advantage of virtual machines, as they provide suitable mechanisms for web browsers testing, as well as for performing a forensic analysis of browsing data. In addition to virtual machines process monitoring is the key concept that has been used

for implementing the testing and forensic analysis framework.
The testing framework aims to automate web browsers, in order to filter and log those events that a web browser has triggered on the local file system, when calling various web sites. The forensic analysis framework processes these log files and performs file recovery from virtual machine disk images. Additionally to file recovery, another benefit of the proposed forensic analysis approach has been the effectiveness of finding valuable private browsing artefacts by means of Digital Forensics XML files.
The evaluation of the test results has been structured as follows: firstly, the manual test scenarios - for both the public and the private mode of the Internet Explorer, Chrome, Firefox, Safari and Opera - have been analysed. Secondly, the automated test results generated by the Internet Explorer's and Chrome's private mode have been inspected, in order to find digital evidence regarding the tested web sites.
The main research question whether there are differences between the manual and the automatic execution of the private mode can be answered as follows: There exist differences between the manual and automated test runs depending on the web sites as well as on the web browsers which have been used for testing.
The evaluation of the test results has shown that the private browsing modes of the tested web browsers have been implemented sufficiently, except that one which is implemented in the Internet Explorer and that one in Safari. Therefore, regarding user privacy, one is recommended to use either Chrome, Firefox or Opera instead of the Internet Explorer and Safari in the tested version.

# Bibliography

[1] Alexa Top 500 Global Sites.
Available: http://www.alexa.com/topsites.
Last accessed: 2013-06-28.

[2] Browser Statistics.
Available: http://www.w3schools.com/browsers/browsers_stats.asp.
Last accessed: 2013-06-28.

[3] Capabilities (aka ChromeOptions) - ChromeDriver.
Available: https://sites.google.com/a/chromium.org/chromedriver/capabilities.
Last accessed: 2013-12-27.

[4] ChromeDriver - selenium - Information about the Chrome Driver - Browser automation framework - Google Project Hosting.
Available: http://code.google.com/p/selenium/wiki/ChromeDriver.
Last accessed: 2013-12-27.

[5] DesiredCapabilities - selenium - (still under work) A specification of DesiredCapabilities and their content when used in JsonWireProtocol - Browser automation framework - Google Project Hosting.
Available: http://code.google.com/p/selenium/wiki/DesiredCapabilities.
Last accessed: 2013-12-27.

[6] DFXML Schema.
Available: https://github.com/dfxml-working-group/dfxml_schema/blob/master/dfxml.xsd.

Last accessed: 2013-06-28.

[7] File Management Functions - (Windows).
Available: http://msdn.microsoft.com/en-us/library/windows/desktop/aa364232(v=vs.85).aspx.
Last accessed: 2013-06-28.

[8] ICAT(1) manual page.
Available: http://www.sleuthkit.org/sleuthkit/man/icat.html.
Last accessed: 2013-06-28.

[9] IFIND(1) manual page.
Available: http://www.sleuthkit.org/sleuthkit/man/ifind.html.
Last accessed: 2013-06-28.

[10] InternetExplorerDriver - selenium - Everything you wanted to know about the Internet Explorer driver - Browser automation framework - Google Project Hosting.
Available: http://code.google.com/p/selenium/wiki/InternetExplorerDriver.
Last accessed: 2013-12-27.

[11] Process Monitor.
Available: http://technet.microsoft.com/de-de/sysinternals/bb896645.aspx.
Last accessed: 2013-06-28.

[12] Selenium - Web Browser Automation.
Available: http://www.seleniumhq.org/.
Last accessed: 2013-06-28.

[13] Selenium WebDriver.
Available: http://www.seleniumhq.org/projects/webdriver/.
Last accessed: 2013-06-28.

[14] The Sleuth Kit (TSK) & Autopsy: Open Source Digital Forensics Tools.
Available: http://www.sleuthkit.org.
Last accessed: 2013-06-28.

[15] WebDriver: Advanced Usage - Selenium Documentation.
Available: http://docs.seleniumhq.org/docs/04_webdriver_advanced.jsp.
Last accessed: 2013-12-27.

[16] ZwSetInformationFile routine (Windows Drivers).
Available: http://msdn.microsoft.com/en-us/library/windows/hardware/ff567096
Last accessed: 2013-12-27.

[17] *Oracle VM VirtualBox: User Manual*, 2013. Version 4.2.14.

[18] G. Aggarwal, E. Bursztein, C. Jackson, and D. Boneh. An analysis of private browsing modes in modern browsers. In *Proceedings of the 19th USENIX Security Symposium*, pages 79–94. USENIX Association, 2010.

[19] M. K. Ahmed, M. Hussain, and A. Raza. An automated user transparent approach to log web urls for forensic analysis. In *Proceedings of the 5th International Conference on IT Security Incident Management and IT Forensics*, pages 120–127. IEEE, 2009.

[20] N. Al Mutawa, I. Al Awadhi, I. Baggili, and A. Marrington. Forensic artifacts of facebook's instant messaging service. In *Proceedings of the 2011 International Conference for Internet Technology and Secured Transactions*, pages 771–776. IEEE, 2011.

[21] A. Alkaabi, G. Mohay, A. McCullagh, and N. Chantler. Dealing with the problem of cybercrime. In *Digital Forensics and Cyber Crime*, pages 1–18. Springer, 2011.

[22] C. Altheide and H. Carvey. *Digital Forensics with Open Source Tools*. Syngress, 1st edition, 2011.

[23] L. Aronson and J. van den Bos. Towards an engineering approach to file carver construction. In *Proceedings of the 35th Annual IEEE Computer Software and Applications Conference Workshops*, pages 368–373. IEEE, 2011.

[24] A. Baláž and R. Hlinka. Forensic analysis of compromised systems. In *Proceedings of the 10th IEEE International Conference on Emerging eLearning Technologies and Applications*, pages 27–30. IEEE, 2012.

[25] J. Bang, B. Yoo, J. Kim, and S. Lee. Analysis of time information for digital investigation. In *Proceedings of the Fifth International Joint Conference on INC, IMS, and IDC*, pages 1858–1864. IEEE, 2009.

[26] D. Barrett and G. Kipper. *Virtualization and Forensics: A Digital Forensic Investigator's Guide to Virtual Environments*. Syngress, 1st edition, 2010.

[27] H. Berghel. Hiding data, forensics, and anti-forensics. *Communications of the ACM*, 50(4):15–20, 2007.

[28] H. Berghel and N. Brajkovska. Wading into alternate data streams. *Communications of the ACM*, 47(4):21–27, 2004.

[29] H. Berghel and D. Hoelzer. Disk wiping by any other name. *Communications of the ACM*, 49(8):17–21, 2006.

[30] D. Brezinski and T. Killalea. Guidelines for evidence collection and archiving. Available: https://www.ietf.org/rfc/rfc3227.txt, 2002. Last accessed: 2013-06-28.

[31] C. L. T. Brown. *Computer Evidence: Collection and Preservation*. Cengage Learning, 2nd edition, 2009.

[32] Y. Cai and Y. Yuan. A comparative study of the safety between internet explorer and firefox. In *Proceedings of the 2012 International Symposium on Information Science and Engineering*, pages 165–168. IEEE, 2012.

[33] B. Carrier. *File System Forensic Analysis*. Addison-Wesley Professional, 1st edition, 2005.

[34] H. Carvey. *Windows Forensic Analysis: DVD Toolkit*. Syngress, 2nd edition, 2009.

[35] H. Carvey. *Windows Registry Forensics: Advanced Digital Forensic Analysis of the Windows Registry*. Syngress, 1st edition, 2011.

[36] H. Carvey. *Windows Forensic Analysis Toolkit: Advanced Analysis Techniques for Windows 7*. Syngress, 3rd edition, 2012.

[37] E. Casey. *Digital Evidence and Computer Crime: Forensic Science, Computer and the Internet*. Academic Press, 3rd edition, 2011.

[38] E. Casey, C. Altheide, C. Daywalt, A. de Donno, D. Forte, J. O. Holley, A. Johnston, R. van der Knijff, A. Kokocinski, P. H. Luehr, T. Maguire, R. D. Pittman, C. W. Rose, D. Shaver, J. J. Schwerha, and J. Reust Smith. *Handbook of Digital Forensics and Investigation*. Academic Press, 1st edition, 2009.

[39] E. Casey and G. J. Stellatos. The impact of full disk encryption on digital forensics. *ACM SIGOPS Operating Systems Review*, 42(3):93–98, 2008.

[40] K. Chang, G. Kim, K. Kim, and W. Kim. Initial case analysis using windows registry in computer forensics. In *Proceedings of the 2007 International Conference on Future Generation Communication and Networking*, pages 564–569. IEEE, 2007.

[41] H. Chivers. Private browsing: A window of forensic opportunity. *Digital Investigation*, In Press(Corrected Proof):Available online 19 December 2013, 2013.

[42] K. P. Chow, F. Y. W. Law, M. Y. K. Kwan, and P. K. Y. Lai. The rules of time on ntfs file system. In *Proceedings of the Second International Workshop on Systematic Approaches to Digital Forensic Engineering*, pages 71–85. IEEE, 2007.

[43] K. Dahbur and B. Mohammad. The anti-forensics challenge. In *Proceedings of the 2011 International Conference on Intelligent Semantic Web-Services and Applications*, pages 14:1–14:7. ACM, 2011.

[44] L. Daniel and L. Daniel. *Digital Forensics for Legal Professionals: Understanding Digital Evidence From the Warrant to the Courtroom*. Syngress, 1st edition, 2011.

[45] J. Davis, J. MacLean, and D. Dampier. Methods of information hiding and detection in file systems. In *Proceedings of the Fifth International Workshop on Systematic Approaches to Digital Forensic Engineering*, pages 66–69. IEEE, 2010.

[46] S. M. Diesburg and A.-I. A. Wang. A survey of confidential data storage and deletion methods. *ACM Computing Surveys*, 43(1):2:1–2:37, 2010.

[47] X. Ding and H. Zou. Time based data forensic and cross-reference analysis. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 185–190. ACM, 2011.

[48] EC-Council. *Computer Forensics: Evidence Collection & Preservation*. Cengage Learning, 1st edition, 2009.

[49] EC-Council. *Computer Forensics: Investigating Network Intrusions & Cybercrime*. Cengage Learning, 1st edition, 2009.

[50] D. Farmer and W. Venema. *Forensic Discovery*. Addison-Wesley Professional, 1st edition, 2005.

[51] S. L. Garfinkel. Carving contiguous and fragmented files with fast object validation. *Digital Investigation*, 4 Supplement:2–12, 2007.

[52] S. L. Garfinkel. Automating disk forensic processing with sleuthkit, xml and python. In *Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering*, pages 73–84. IEEE, 2009.

[53] S. L. Garfinkel. Digital forensics xml and the dfxml toolset. *Digital Investigation*, 8(3-4):161–174, 2012.

[54] G. Giustini, M. Andreolini, and M. Colajanni. Open source live distributions for computer forensics. In *Open Source Software for Digital Forensics*. Springer, 2010.

[55] H. Guo and M. Xu. A method for recovering jpeg files based on thumbnail. In *Proceedings of the 2011 International Conference on Control, Automation and Systems Engineering*, pages 1–4. IEEE, 2011.

[56] R. Guo, T. Cao, and X. Luo. Application layer information forensics based on packet analysis. In *Proceedings of the 2010 International Conference of Information Science and Management Engineering (ISME)*, volume 1, pages 206–209. IEEE, 2010.

[57] Y. Guo and J. Slay. A function oriented methodology to validate and verify forensic copy function of digital forensic tools. In *Proceedings of the Fifth International Conference on Availability, Reliability, and Security*, pages 665–670. IEEE, 2010.

[58] C. Haiping, L. Delin, G. Qinquan, Q. Zhicong, and W. Shunxiang. Ie internet information forensics technology in unallocated disk space. In *Proceedings of the First International Symposium on Computer Network and Multimedia Technology*, pages 1–4. IEEE, 2009.

[59] D. Hayes and S. Qureshi. Implications of microsoft vista operating system for computer forensics investigations. In *Proceedings of the 2009 Long Island Systems, Applications and Technology Conference*, pages 1–9. IEEE, 2009.

[60] D. Hayes, V. Reddy, and S. Qureshi. The impact of microsoft's windows 7 on computer forensics examinations. In *Proceedings of the 2010 Long Island Systems Applications and Technology Conference*, pages 1–6. IEEE, 2010.

[61] K. J. Jones, R. Bejtlich, and C. W. Rose. *Real Digital Forensics: Computer Security and Incident Response*. Addison-Wesley Professional, 1st edition, 2005.

[62] D. Kahvedžić and T. Kechadi. On the persistence of deleted windows registry data structures. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 895–896. ACM, 2009.

[63] M. D. Kelly and S. J. Geoghegan. First forensic internet replay sequencing tool. In *Proceedings of the Eighth IEEE International Symposium on Network Computing and Applications*, pages 270–273. IEEE, 2009.

[64] K. Kent, S. Chevalier, T. Grance, and H. Dang. *Guide to Integrating Forensic Techniques into Incident*. National Institute of Standards and Technology, 2006.

[65] B. Li, L. Wang, Y. Sun, and Q. Wang. Image fragment carving algorithms based on pixel similarity. In *Proceedings of the 2012 Fourth International Conference on Multimedia Information Networking and Security*, pages 979–982. IEEE, 2012.

[66] T. V. Lillard, C. P. Garrison, C. A. Schiller, and J. Steele. *Digital Forensics for Network, Internet, and Cloud Computing: A Forensic Evidence Guide for Moving Targets and Data*. Syngress, 1st edition, 2010.

[67] S. Lim, J. Park, K. Lim, C. Lee, and S. Lee. Forensic artifacts left by virtual disk encryption tools. In *Proceedings of the 2010 3rd International Conference on Human-Centric Computing*, pages 1–6. IEEE, 2010.

[68] Z. Liu, Y. Chen, W. Yu, and X. Fu. Generic network forensic data acquisition from household and small business wireless routers. In *Proceedings of the 2010 IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks*, pages 1–6. IEEE, 2010.

[69] A. Marrington, I. Baggili, T. A. Ismail, and A. A. Kaf. Portable web browser forensics: A forensic examination of the privacy benefits of portable web browsers. In *Proceedings of the 2012 International Conference on Computer Systems and Industrial Informatics*, pages 1–6. IEEE, 2012.

[70] A. I. Martini, A. Zaharis, and C. Ilioudis. Detecting and manipulating compressed alternate data streams in a forensics investigation. In *Proceedings of the Third International Annual Workshop on Digital Forensics and Incident Analysis*, pages 53–59. IEEE, 2008.

[71] K. M. Mohamad, A. Patel, and M. M. Deris. Carving jpeg images and thumbnails using image pattern matching. In *Proceedings of the 2011 IEEE Symposium on Computers & Informatics*, pages 78–83. IEEE, 2011.

[72] L. Naiqi, W. Zhongshan, H. Yujie, and QinKe. Computer forensics research and implementation based on ntfs file system. In *Proceedings of the 2008 ISECS International Colloquium on Computing, Communication, Control, and Management*, volume 1, pages 519–523. IEEE, 2008.

[73] B. Nelson, A. Phillips, and C. Steuart. *Guide to Computer Forensics and Investigation*. Cengage Learning, 4th edition, 2009.

[74] D. J. Ohana and N. Shashidhar. Do private and portable web browsers leave incriminating evidence? In *Proceedings of the 2013 IEEE Security and Privacy Workshops*, pages 135–142. IEEE, 2013.

[75] F. Olajide, N. Savage, G. Akmayeva, and C. Shoniregun. Digital forensic research - the analysis of user input on volatile memory of windows application. In *Proceedings of the 2012 World Congress on Internet Security*, pages 231–238. IEEE, 2012.

[76] F. Olajide, N. Savage, G. Akmayeva, and R. Trafford. Forensic memory evidence of windows application. In *Proceedings of the 2012 International Conference for Internet Technology and Secured Transactions*, pages 715–718. IEEE, 2012.

[77] A. Pal and N. Memon. The evolution of file carving. *IEEE Signal Processing Magazine*, 26(2):59–71, 2009.

[78] G. Pecherle, C. Gyorodi, R. Gyorodi, B. Andronic, and I. Ignat. New method of detection and wiping of sensitive information. In *Proceedings of the 2011 IEEE 7th International Conference on Intelligent Computer Communication and Processing*, pages 145–148. IEEE, 2011.

[79] A. Philipp, D. Cowen, and C. Davis. *Hacking Exposed Computer Forensics: Secrets & Solutions*. McGraw-Hill Osborne Media, 2nd edition, 2009.

[80] K. Rankin. Hack and /: remotely wipe a server. *Linux Journal*, 2011(209):11:1–11:4, 2011.

[81] M. Reith, C. Carr, and G. Gunsch. An examination of digital forensic models. *International Journal of Digital Evidence*, 1(3):1–12, 2002.

[82] M. K. Rogers, J. Goldman, R. Mislan, T. Wedge, and S. Debrota. Computer forensics field triage process model. *Journal of Digital Forensics, Security and Law*, 1(2):1–38, 2006.

[83] V. Roussev and S. L. Garfinkel. File fragment classification-the case for specialized approaches. In *Proceedings of the Fourth International Workshop on Systematic Approaches to Digital Forensic Engineering*, pages 3–14. IEEE, 2009.

[84] M. E. Russinovich and A. Margosis. *Windows Sysinternals Administrator's Reference*. O'Reilly Media, Inc., 1st edition, 2011.

[85] M. E. Russinovich, D. A. Solomon, and A. Ionescu. *Windows Internals, Part 1: Covering Windows Server 2008 R2 and Windows 7*. Microsoft Press, 6th edition, 2012.

[86] H. Said, N. Al Mutawa, I. Al Awadhi, and M. Guimaraes. Forensic analysis of private browsing artifacts. In *Proceedings of the 2011 International Conference on Innovations in Information Technology*, pages 197–202. IEEE, 2011.

[87] J. Sammons. *The Basics of Digital Forensics: The Primer for Getting Started in Digital Forensics*. Syngress, 1st edition, 2012.

[88] J. Schlamp, G. Carle, and E. W. Biersack. A forensic case study on as hijacking: the attacker's perspective. *ACM SIGCOMM Computer Communication Review*, 43(2):5–11, 2013.

[89] D. L. Shinder and M. Cross. *Scene of the Cybercrime*. Syngress, 2nd edition, 2008.

[90] G. Sibiya, H. S. Venter, S. Ngobeni, and T. Fogwill. Guidelines for procedures of a harmonised digital forensic process in network forensics. In *Proceedings of the Information Security for South Africa*, pages 1–7. IEEE, 2012.

[91] M. C. Stamm and K. J. R. Liu. Anti-forensics for frame deletion/addition in mpeg video. In *Proceedings of the 2011 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1876–1879. IEEE, 2011.

[92] M. C. Stamm, S. K. Tjoa, W. S. Lin, and K. J. R. Liu. Undetectable image tampering through jpeg compression anti-forensics. In *Proceedings of the 2010 IEEE International Conference on Image Processing*, pages 2109–2112. IEEE, 2010.

[93] M. C. Stamm, M. Wu, and K. J. R. Liu. Information forensics: An overview of the first decade. *IEEE Access*, 1:167–200, 2013.

[94] I. Sutherland, J. Evans, T. Tryfonas, and A. Blyth. Acquiring volatile operating system data tools and techniques. *ACM SIGOPS Operating Systems Review*, 42(3):65–73, 2008.

[95] Z. Tang, H. Ding, M. Xu, and J. Xu. Carving the windows registry files based on the internal structure. In *Proceedings of the 2009 First International Conference on Information Science and Engineering*, pages 4788–4791. IEEE, 2009.

[96] Y. Wei, N. Zheng, and M. Xu. An automatic carving method for rar file based on content and structure. In *Proceedings of the Second International Conference on Information Technology and Computer Science*, pages 68–72. IEEE, 2010.

[97] H. Xie, K. Jiang, X. Yuan, and H. Zeng. Forensic analysis of windows registry against intrusion. *International Journal of Network Security & Its App lications*, 4(2):121–134, 2012.

[98] B. Yoo, J. Park, J. Bang, and S. Lee. A study on a carving method for deleted ntfs compressed files. In *Proceedings of the 2010 3rd International Conference on Human-Centric Computing*, pages 1–6. IEEE, 2010.

[99] S. Zhang, L. Wang, and L. Zhang. Extracting windows registry information from physical memory. In *Proceedings of the 2011 3rd International Conference on Computer Research and Development*, volume 2, pages 85–89. IEEE, 2011.

[100] S. Zhang, L. Wang, R. Zhang, and Q. Guo. Exploratory study on memory analysis of windows 7 operating system. In *Proceedings of the 2010 3rd International Conference on Advanced Computer Theory and Engineering*, volume 6, pages V6–373–V6–377. IEEE, 2010.