JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

Technisch-Naturwissenschaftliche
Fakultät

Automatische Web-History Analyse

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Masterstudium

Netzwerke und Sicherheit

Eingereicht von:
Gerald Prock, Bakk.

Angefertigt am:
Institut für Informationsverarbeitung und Mikroprozessortechnik - FIM

Beurteilung:
Assoz.Prof. Priv.-Doz. Mag. iur. Dipl.Ing. Dr. Michael Sonntag

Linz, Dezember 2010

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplom- bzw. Magisterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum

.....

Unterschrift

.....
(Gerald Prock, Bakk.)

Kurzfassung

Diese Arbeit stellt ein Programm zum automatischen Auswerten und Validieren von Browser-History-Daten aus einem Festplattenimage vor.

Bei dem Image kann es sich um ein RAW-Image einer Partition oder einer kompletten Festplatte handeln. Dieses wird von der erstellten Anwendung in das Betriebssystem nur lesend eingebunden und anschließend auf Benutzer und Browser hin untersucht. Unterstützt werden dabei folgende Browser:

- Firefox von Mozilla ab Version 3
- Opera von Opera ab Version 6
- Chrome von Google ab Version 1
- InternetExplorer von Microsoft aber Version 5

Ist die gesamte Datenquelle durchsucht, werden die gefundenen Ordner in ein temporäres Verzeichnis kopiert und deren Web-History Inhalt ausgewertet. Dabei werden auch der jeweilige Name des zugehörigen Benutzers und der jeweilige Typ des Betriebssystem mit gespeichert.

Sämtliche gefundenen URLs und Cookies werden anschließend mehreren Tests unterzogen, um die Validität zu garantieren. Dazu werden die entsprechenden Adressen heruntergeladen und mit den lokalen Daten verglichen.

Die gefundenen Daten werden dabei in einer SQLite-Datenbank gespeichert und diese wird abschließend als XML-Datei exportiert. Die Hierarchie dieser Datei kann dabei direkt beim Start der Anwendung eingestellt werden.

Das erstellte Programm wurde in Bash geschrieben und ist auf jedem GNU|Linux ausführbar. Zusätzlich wurde es auf einer Live-CD installiert, die dieser Arbeit beigelegt ist. Dadurch können auch Computer ohne das Ausbauen der Festplatte und unabhängig vom installierten Betriebssystem untersucht werden.

Abstract

This work is about a created program, which can automatically evaluate and validate browser history data from a disk image.

The mentioned image can be a RAW-image of a partition or an entire hard disk. This will be read only mounted from the created program and after that analyzed for the existing user and browsers. The following browser are supported:

- Firefox by Mozilla from version 3
- Opera by Opera from version 6
- Chrome by Google from version 1
- InternetExplorer by Microsoft from version 5

Is the search through the data source complete, then the found browser history directories will be copied to a temporary folder. After that all files will be evaluated and also the according user and operating system type will be saved.

All found URL's and cookies will be checked for their validation through downloading of the corresponding address and comparing them with the local data.

The found data will be stored in a SQLite-database and at the end of the program a XML-file will be created out of it. The hierarchy of this file can be defined at the start of the application.

The created program was written in Bash and can be executed on any GNU|Linux operating system. In addition a Live-CD with the program was created and enclosed to this work. This enables to review a computer without removing the harddisk or starting the operating system.

Inhaltsverzeichnis

Abbildungsverzeichnis

Die Bilder wurden, sofern nicht anders angegeben, vom Autor selbst erstellt.

Änderungen oder Anpassungen von Bildern aus externen Quellen werden durch einen zusätzlichen Hinweis bei der Zitierung kenntlich gemacht.

Abkürzungen

In diesem Kapitel werden die im Laufe der Arbeit erwähnten Abkürzungen beschrieben:

- **Linux** ... Unter diesem Begriff ist in dieser Arbeit das Betriebssystem GNU|Linux gemeint, denn der Begriff Linux selbst steht eigentlich nur für den Kernel des Systems ohne weitere Programme.

- **GNU** ... Diese Abkürzung steht für das Projekt "GNU is not Unix" und hat die Zielsetzung, ein freies Betriebssystem in der Art von Unix zu erstellen. Als Kernel wird dabei Linux genutzt. Sämtliche Anwendungen des Projektes stehen unter der GPL-Lizenz.

- **Kernel** ... Dabei handelt es sich um den Teil eines Betriebssystems, der für die Ansteuerung der Hardware, Verteilung der Ressourcen und die Ausführung der Programme zuständig ist, also den zentralsten Teil eines jeden Betriebssystems.

- **GPL** ... Dies ist die Abkürzung für die OpenSource-Lizenz "General Public License", unter welcher auch der Linux-Kernel steht. Diese erlaubt es den Quellcode frei zu nutzen und zu verändern. Doch diese Änderungen müssen jedoch ebenfalls wieder freigegeben werden. Diese Art von Lizenz wird auch als "Copyleft" bezeichnet.

- **BSD** ... Hierbei handelt es sich um ein Betriebssystem, welches ähnlich wie GNU|Linux auf der Basis von Unix aufgebaut ist und sich stark an dessen Richtlinien orientiert.

- **OsX** ... Dieser Begriff steht für das Betriebssystem von Apple. Der vollständige Name lautet "Mac OS X". Die Grundlagen des Systems sind an Unix und BSD angelehnt.

- **Bash** ... Dabei handelt es sich um die bekannteste Kommandozeile von den GNU-Tools, die sehr viele Möglichkeiten zur Erstellung von Skripten beinhaltet.

- **Skript** ... Unter diesem Begriff versteht man ein Programm, welches ohne Übersetzung in Maschinensprache oder Bytecode direkt ausgeführt werden kann. Für die Bearbeitung von Skripten wird nur ein Texteditor benötigt.

- **URL** ... Dabei handelt es sich um eine vollständige Adresse eines Netzwerkprotokolls. Die Abkürzung steht für "Uniform Resource Locator" und wird meistens in Verbindung mit HTTP gebracht. Andere Protokolle wie beispielsweise FTP oder Webdav können ebenfalls angesprochen werden.
- **HTTP** ... Dabei handelt es sich um das "Hypertext Transfer Protocol", welches häufig zur Übertragung von HTML-Seiten oder Dateien vom Server zu einem Webbrowser eingesetzt wird. Daten können allerdings damit in beide Richtungen transportiert werden.
- **HDD** ... Diese Abkürzung steht für eine Computer-Festplatte ("Hard Disk Drive").
- **NTFS** ... Unter diesem Begriff versteht man das aktuelle Dateisystem von Microsoft Windows, welches sich mit vollem Namen "New Technology File System" nennt.
- **FAT** ... Diese Abkürzung steht für "File Allocation Table" und bezeichnet ein Dateisystem, das früher von Microsoft Windows (bis Windows 2000) genutzt wurde. Heute ist es noch oft auf Speichermedien zu finden.
- **EXT** ... Dabei handelt es sich um ein häufig genutztes Dateisystem von Linux. Die Abkürzung steht für "extended filesystem" und existiert momentan in mehreren Versionen, wobei "ext4" die neuere Version ist.
- **RAW** ... Dieser Begriff steht für Rohdaten und bedeutet in Bezug auf eine Festplatte, dass kein Dateisystem vorhanden ist. Dateien können aber sehr wohl vorhanden sein. Allerdings gibt es keine sofort auslesbare Zuordnung, wodurch eigene Programme zum Auffinden benötigt werden.
- **ZIP** ... Dabei handelt es sich um ein Dateiformat für komprimierte Archiv-Dateien. Das Format selbst ist quelloffen und es existieren Implementierungen für alle aktuellen Betriebssysteme.
- **ID** ... Diese Abkürzung steht für "Identifikator" und wird gerne für die eindeutige Identifizierung oder Zuordnung von Daten genutzt.
- **Live-CD** ... Unter diesem Begriff versteht man ein Betriebssystem, welches direkt von einem Wechseldatenträger auf einem beliebigen Computer gestartet werden kann. Häufig handelt es sich dabei um ein GNU|Linux-Betriebssystem.

1 Danksagung

Bedanken möchte ich mich sehr herzlich bei allen Professoren und Instituten, die mich bei meinem Studium in den vergangenen Semestern unterstützt haben.

Dies gilt besonders für das Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM) und meinem Betreuer Herrn Michael Sonntag, der mir eine konkrete Aufgabe gestellt hat und mir die Gelegenheit gegeben hat, diese selbstständig zu verwirklichen.

Des weiteren möchte ich mich bei den Entwicklern der Linuxdistribution von aptosid (früher sidux) für die Unterstützung in Form von Hinweisen bedanken. Dies gilt auch für die Tipps zum Bauen der Live-CD.

Bedanken möchte ich mich außerdem bei meinen Eltern, die mir das Studium finanziell ermöglicht haben.

2 Einleitung

Das Ziel dieser Arbeit ist es eine Vielzahl von Programmen zu entwickeln, welche Webbrowser-History-Daten aus einem Festplattenimage automatisiert auswertet und in ein weiterverarbeitbares Format bringen. Genauere Informationen zur Aufgabenstellung sind in Kapitel Zielsetzung auf Seite 32 zu finden.

In der heutigen Zeit findet ein Großteil des weltweiten Informationsflusses über das Internet statt. Auch innerhalb von vielen Firmen werden Informationen über ein eigenes Intranet bereitgestellt und können von den Angestellten abgerufen werden. Durch die aktuellen Begriffe wie Web2.0 oder HTML5 wird außerdem der Web-Browser immer mehr zum zentral genützten Programm auf einem Arbeits- oder Privatrechner.

Daher ist es auch nicht verwunderlich, dass der Browser zu einem beliebten Programm für nicht legale Tätigkeiten im Netz wurde. Um solche Rechtsverletzungen nachzuweisen, wird geschultes Personal benötigt, welches den Rechner forensisch untersucht und gefundene Spuren als Beweis sicherstellt und dokumentiert.
Viele Privatpersonen wollen außerdem zu Recht wissen, welche Tätigkeiten ihr Rechner im Hintergrund durchführt und welche Daten über das Netzwerk übertragen werden. Daher ist es auch notwendig zu wissen, welche Seiten ein Web-Browser zu welchem Zeitpunkt besucht hat.

Das im Zuge der Masterarbeit entwickelte Programm unterstützt sowohl Forensiker als auch Privatpersonen bei der Analyse von Browserdaten. Unterstützt werden dabei die nachfolgend aufgelisteten bekanntesten Browser für GNU|Linux und Windows. Genauere Informationen zu den jeweiligen Browsern sowie deren Auslesemöglichkeiten sind im Kapitel Browserforensic auf Seite 17 zu finden.

- Mozilla Firefox
  (und alle direkt darauf basierenden Browser wie beispielsweise Iceweasel)
- Opera
- Google Chrome
  (und alle direkt darauf basierenden Browser wie etwa Chromium)
- Microsoft Internet Explorer

Der Aufbau dieser Arbeit ist folgender:

Im nachfolgenden Kapitel 3 auf der nächsten Seite wird auf die Browserforensik mit deren Möglichkeiten, Pflichten und Problemen eingegangen.

Anschließend werden in Kapitel 4 auf Seite 17 die bereits angesprochenen Browser behandelt.

Im Anschluss daran werden in Kapitel 5 auf Seite 32 die exakte Zielsetzung sowie zusätzliche im Vorhinein geplante Zusatzpunkte besprochen.

In Kapitel 6 auf Seite 35 wird das erstellte Programm erläutert. Dort gibt es neben der Besprechung des Aufbaus auch Hinweise zur Programmierung. Ab dem Unterpunkt 6.7 auf Seite 60 sind außerdem Informationen zum jeweiligen Quellcode zu finden.

Im nachfolgenden Kapitel 7 auf Seite 104 wird besprochen, welche Funktionen die erstellte Anwendung bietet und wie diese bedient und installiert werden kann. Zudem wird auf deren Vor- und Nachteile eingegangen und bereits durchgeführte Tests werden aufgelistet.

In Kapitel 8 auf Seite 125 sind die eigenen Gesichtspunkte und weitere geplante Schritte für das Programm zu finden.

3 Grundlagen der Browserforensik

Die Browserforensik befasst sich mit der Analyse von gespeicherten Browserdaten auf einem Datenträger. Dabei werden so viele Informationen wie möglich aus den zu analysierenden History-Dateien gewonnen, ohne diese in irgendeiner Weise zu verändern. Dadurch können unter anderem unwiderlegbare Beweise erstellt werden, die auch vor Gericht Gültigkeit besitzen. Auslesbare Daten sind beispielsweise die besuchten Webadressen inklusive der jeweiligen Uhrzeit.

Vorab ist zu erwähnen, dass bei der Recherche der alternativen Arbeiten keine gleichwertige Applikation gefunden werden konnten. Zwar existieren einige alternative Programme, jedoch keines arbeitet vollautomatisiert von einem anderen Betriebssystem aus. Genaueres dazu ist in Kapitel 3.5 auf Seite 9 nachzulesen.

3.1 Möglichkeiten

Mit Hilfe die Browserforensik ergibt sich die Möglichkeit nachträglich zu prüfen, welche Informationen der jeweilige Browser auf der Festplatte hinterlässt. Durch diese Informationen können beispielsweise Forensiker feststellen, welche Adressen der Benutzer des Browsers zu welchem Zeitpunkt besucht hat. Es ist aber auch interessant zu sehen, welche zusätzlichen URLs ein Browser im Hintergrund automatisiert aufgerufen hat. Die auslesbaren Daten, die für das im Zuge dieser Arbeit erstellte Programm genutzt werden, sind in den jeweiligen Unterpunkten von Kapitel 4 auf Seite 17 zu finden.

Ausgelesen werden können etwa die exakte URL, die Besuchszeit, die gesetzten Cookies, die zugehörigen Cache-Dateien. Mit entsprechenden Tools ist es auch möglich, alle im Cache gespeicherten Webseiten lokal zu öffnen. Mögliche Programme dafür sind im Kapitel 3.5 auf Seite 9 zu finden. Außerdem können diese Dateien auch teilweise händisch entpackt und betrachtet werden [1].

Der jeweilige Benutzer kann zwar die gespeicherten Dateien beeinflussen, zum Beispiel löschen, dennoch können die Informationen teilweise noch nachträglich gewonnen werden. Nachfolgend wird erwähnt, welche Möglichkeiten es dafür gibt.

Zum Schutz der Privatsphäre können die History-Dateien vom jeweiligen Browser gelöscht werden. Jedoch mit speziellen Recovery-Programmen, wie beispielsweise “photorec”, lassen sich diese teilweise wiederherstellen. Dies funktioniert vor allem dann, wenn der betreffende Bereich auf der Festplatte nicht von neuen Daten überschrieben wurde. Überschriebene History-Dateien lassen sich nicht mehr wiederherstellen.

Der sogenannte “Stealth-Modus”, den viele Browser anbieten, hinterlässt ebenfalls Spuren. In diesem Modus werden möglichst wenig Informationen zum jeweiligen Webserver übertragen und bei Beendigung der Cache automatisch geleert. Jedoch können diese Dateien ebenfalls auf Dateisystembasis mit dem besagten Programm wiederhergestellt werden.

Den kompletten Browserordner in den Arbeitsspeicher zu legen und somit ein Anlegen von Dateien auf der Festplatte zu verhindern, funktioniert nur dann zuverlässig, wenn kein virtueller Speicher (SWAP oder Auslagerungsdatei) existiert. Denn dort gespeicherte Informationen werden ebenfalls von den Datei-Recovery-Programm gefunden.

Bei allen wiederhergestellten Dateien gibt es allerdings ein Problem bei der Zuordnung zu einem Benutzer. Es kann zwar festgestellt werden, dass die gefundenen URLs vom jeweiligen System aus besucht wurden, eine genaue Zuordnung bei Mehrbenutzersystemen ist allerdings nur mehr schwer möglich.

Es kann daher zusammenfassend festgehalten werden:

> Jede History-Datei, welche auf der Festplatte angelegt wurde, ist auslesbar, solange sie nicht überschrieben wurde. Die Zuordnung zu einem speziellen Benutzer kann allerdings verschleiert werden.

Ein Problem bei sämtlichen forensischen Untersuchungen besteht darin, dass es mittlerweile viele verschiedene Browser gibt. Aus diesem Grunde beschränkt sich diese Arbeit auf die vier am meisten verbreiteten Browser, nämlich Firefox, Opera, Chrome und InternetExplorer. Einige alternative Browser nutzen Firefox oder Chrome als Grundgerüst, weshalb der Cache auf der Festplatte identisch ist und daher auch ausgelesen werden kann. Beispiele dafür sind Flock [2] oder RockMelt [3]. Diese werden zwar erkannt, aber als Browser der jeweiligen Abstammung bezeichnet. Alternative exotische Browser wie beispielsweise eLinks [4] wurden aufgrund der sehr geringen Verbreitung ignoriert.

3.2 Anwendungsgebiete

Die Anwendungsgebiete für die Auswertung der Browser-History-Daten sind vielseitig. Die nachfolgende Aufzählung gibt mögliche Beispiele an:

1. **Polizei:**
   Zum Nachweis von strafrechtlichen Delikten, welche im Internet durchgeführt wurden, eignen sich die Browserdaten hervorragend. Vor allem in Verbindung mit eventuellen Aufzeichnungen der Provider oder Serverbetreiber können gültige und unwiderlegbare Beweise erstellt werden. Wichtig dabei ist, dass während der forensischen Untersuchung keine Daten verändert werden. Daher ist vor allem das Eingabeformat von besonderer Bedeutung. Genaueres dazu in Kapitel 3.3 auf der nächsten Seite.

2. **Forschung:**
   Für die Forschung ist das nachträgliche Auswerten der Browser-History ebenfalls interessant. So kann auch ohne Netzwerküberwachung das Verhalten des Browsers beobachtet und getestet werden. Es kann auch festgestellt werden, wie sich manipulierte Webseiten auf den Browser auswirken und ob eventuell ein entfernter Zugriff auf die History-Daten möglich ist. Dadurch können auch eventuelle Sicherheitslücken entdeckt und behoben werden.

3. **Selbstkontrolle:**
   Private Benutzer können ebenfalls ein Interesse an den eigenen Browserdaten haben. Sei es nur zum Überprüfen, ob die eingesetzten Lösch-Tools funktionieren, oder ob die im Browser angegebenen Informationen mit den ausgelesenen identisch sind. Es ist auch interessant zu sehen, welche Webseiten im Hintergrund geladen wurden.

Die angeführten Beispiele sind nur exemplarisch und daher keinesfalls vollständig. Es gibt zudem viele weitere Anwendungen, bei denen die History-Daten genutzt werden können. Dazu gehören unter anderem auch illegale Tätigkeiten, wie das Auslesen der gespeicherten Passwörter und Formulardaten durch Sicherheitslücken im jeweiligen Browser.

3.3 Eingabeformate

Es gibt nur wenig Möglichkeiten, die existierenden History-Daten an ein Programm zu übergeben, ohne dabei Informationen zu verlieren. In der nachfolgenden Auflistung werden mögliche Beispiele angeführt:

- **Übergabe des Orginalverzeichnises:**
  An das zu bearbeitende Programm wird das Browserverzeichnis übergeben. Dieses kann alle darin vorhandenen Informationen auslesen. Probleme gibt es bei dieser Methode dann, wenn sich der Cache in einem anderen Ordner befindet, wie es beim Firefox oder Opera unter Windows der Fall ist. Es müssen auch zusätzlich Vorkehrungen zum Nachweis der Unverfälschtheit, beispielsweise mit Prüfsummen, getroffen werden.

- **Übergabe als Archivdatei:**
  Der komplette Browserordner inklusive zugehörigem Cache kann als Datenarchiv an das Programm übergeben werden. Zur Bearbeitung wird dieses in einem temporären Ordner entpackt und dort verarbeitet. So kann auch jederzeit getestet werden, ob die entpackte Datei noch mit jener aus dem Archiv identisch ist. Zusätzlich werden auch alle Berechtigungen systemübergreifend archiviert.

- **Übergabe als Festplatten- oder Partitionsimage:**
  Eine weitere Möglichkeit ist das Auslesen der Daten über ein Festplattenimage. Wird dieses nur lesend eingebunden, kann mit Prüfsummen jederzeit sichergestellt werden, dass keine Datei verändert wurde. Problematisch ist dabei das Auffinden des eigentlichen Ordners, da sich dieser irgendwo im System befinden kann. Dafür ist aber eine Suche nach gelöschten Dateien möglich, was bei keinen der oben angeführten Möglichkeiten funktioniert.

Ein Vorteil für eine bessere Nachweisbarkeit der Datenintegrität besteht darin, dass wenn die History-Dateien über einen nur lesbaren Datenträger wie beispielsweise einer CD inklusive erstellter Prüfsummen übergeben wird, definitiv sichergestellt werden kann, dass keine Datei während der Verarbeitung geändert wurde.

Die meisten existierenden Programme arbeiten über die Angabe des Ordners oder suchen diesen selbstständig im lokalen System. Beispiele dafür werden in Kapitel 3.5 auf Seite 9

angeführt. Das im Zuge dieser Arbeit erstellte Programm arbeitet mit einem Image und sucht alle dort vorhandenen Verzeichnisse. Genaueres dazu in Kapitel 6 auf Seite 35.

3.4 Ausgabeformate

Im Gegensatz zu den Eingabeformaten kann das Ausgabeformat beliebig gewählt werden. In der nachfolgenden Auflistung werden ein paar gängige Möglichkeiten besprochen:

- **Textdatei:**
  Die Ausgabe kann als formatierte und für Menschen lesbare Textdatei erfolgen. Sollen diese Daten weiterverarbeitet werden, wird ein Parser benötigt, der unter Umständen auch die Formatierung rückgängig machen muss.

- **CSV-Datei:**
  Die gewonnenen Informationen können auch als Textdatei mit Trennzeichen zwischen den einzelnen Werten ausgegeben werden. Üblich sind hier Beistrich, Strichpunkt, Tabulator oder fixe Abstände. Probleme gibt es dann, wenn die enthaltenen Daten ebenfalls dieses Zeichen enthalten. Ist dies nicht der Fall, kann dieses Format einfach weiterverarbeitet werden.

- **XML-Datei:**
  Eine besonders geeignete Variante ist die Ausgabe als XML-Datei, da deren Inhalte von einem Mensch einfach gelesen werden können. Die Weiterverarbeitung etwa in einer Webseite ist ebenfalls gut möglich. Allerdings müssen ein paar Sonderzeichen angepasst werden, damit eine standardisierte Datei erstellt werden kann. Genaueres dazu in Kapitel 6.7.38 auf Seite 97.

- **Tabellendokument:**
  Die Ausgabe des Programmes kann auch als Tabellendokument erfolgen und anschließend, je nach Format, mit Microsoft-Excel, OpenOffice-Calc oder ähnlichen Programmen betrachtet werden. Im Gegensatz zu CSV-Dateien können hier im Wesentlichen alle Zeichen in den Daten vorkommen. Die Sortierung und Auswertung der Daten kann ebenfalls in diesem Format erfolgen.

- **Datenbank:**
  Eine sehr gute Möglichkeit ist die Ausgabe als Datenbank wie beispielsweise SQLite. Darin können alle Zeichen genutzt werden, und es gibt für die Auswertung viele

Abfrage- und Sortiermöglichkeiten. Außerdem kann jedes der anderen erwähnten Formate aus diesem mittels Wrapper erzeugt werden. Dieses Format ist für Menschen allerdings nur mit einem Zusatzprogramm lesbar.

Zusätzlich kann die Ausgabe in einem binären Format erfolgen, wodurch die gewonnenen Informationen nur mehr durch das jeweilige Programm betrachtet werden können. Abhängig von der genutzten Anwendung gibt es oft auch eine eigene Exportfunktion, welche eine der angesprochenen Ausgaben erstellen kann. Genaueres dazu ist im nachfolgenden Kapitel 3.5 bei den jeweiligen Programmen zu finden.

3.5 Existierende Programme

Zum Auswerten der History einzelner Browser gibt es viele verschiedene Programme. Einige davon müssen allerdings auf dem jeweiligen System installiert werden, wodurch diese für forensische Untersuchungen nicht wirklich verwendbar sind. In diesem Kapitel werden einige in der Auflistung der Programme vorgestellt. Darunter befinden sich auch Applikationen, mit denen die Cachedateien geöffnet werden können und deren Inhalt in einem lokalen Browser betrachtet werden kann.

Interessant ist, dass es momentan keine gleichwertige Alternative zu dem im Zuge dieser Arbeit erstellten Anwendung gibt. Es können zwar mit einigen Programmen mehrere Browser von einem anderen System aus verarbeitet werden, jedoch nicht vollautomatisiert von einem Image als Datenquelle. Ein manuelles Einbinden des Image, das Suchen des Browserordners und das anschließende Aufrufen eines der möglichen Programme funktioniert hingegen fast immer.

Außerdem gibt es nur wenige Programme, welche direkt unter Linux installiert oder ausgeführt werden können. Fast alle werden ausschließlich für Microsoft Windows erstellt. Eine Nutzung dieser Programme unter Linux mittels Wine ist zwar möglich, ein vollständiger Funktionsumfang kann jedoch dabei nicht garantiert werden. Allerdings gibt es mehrere Live-CDs [5], welche eine händische Untersuchung eines Rechners oder eines Images ermöglichen. Diese sind ebenfalls in der Auflistung in diesem Kapitel angeführt.

Im Internet existieren zahlreiche Anleitungen zum Auslesen der History-Daten. Ein gutes Beispiel existiert von Symantec, welches mehrere Programme inklusive deren praktische

Anwendbarkeit vorstellt [6]. Die dort aufgeführten Programme sind ebenfalls in der Auflistung zu finden.

- **pasco und galleta** [7]
  Mit dem OpenSource-Programm "pasco", welches für Windows und Linux existiert, können die "index.dat"-Dateien des InternetExplorers in ein für Menschen lesbares Format überführt werden. Dieses Programm ist ein Bestandteil des Odessa-Toolkit. Zu diesem gehört auch "galleta", welches die Cookies des InternetExplorers in ein lesbares Format überführt. Beide Anwendungen können nur von der Befehlszeile aus angesprochen werden, da eine grafische Oberfläche nicht existiert. Die nachfolgende Abbildung zeigt die Ausgabe von "pasco", welche in OpenOffice-Calc importiert wurde.

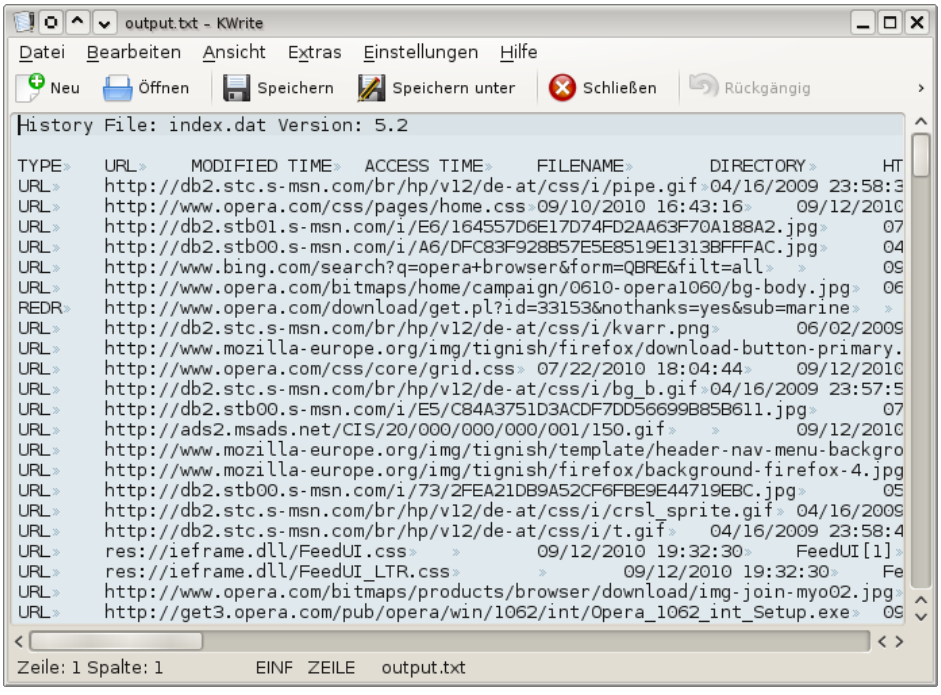
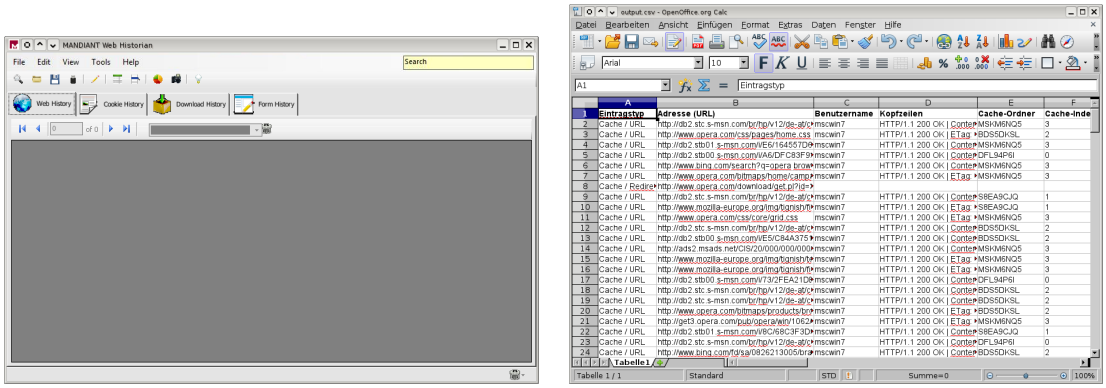


Abbildung 1: Die Ausgabe von Pasco in dem Texteditor "kwrite" geöffnet

- **Web Historian** [8]
  Bei diesem Programm der Firma Mandiant handelt es sich um ein umfangreiches Tool zum Auslesen der History-Daten von verschiedenen Browsern unter Microsoft

Windows. Vertrieben wird es als Freeware-Applikation, die zwar frei verwendet, aber nicht weitergegeben werden darf. Ausgelesen werden können die Browser Firefox ab Version 2, Chrome ab Version 3, InternetExplorer aber Version 5 und Safari ab Version 3. Die Ausgabe erfolgt als XML oder CSV-Datei. Die Anwendung selbst kann nur grafisch bedient werden, eine Version für die Befehlszeile existiert nicht. Ein von Hand eingebundenes Image kann mit diesem Programm verarbeitet werden, wenn die History-Datei oder der Profilordner ebenfalls händisch angegeben wird. Die nachfolgende linken Abbildung zeigt ein mit "Web Historian" erstelltes XLS-Dokument an.



(a) Programmfenster von Web Historian (b) Geöffnete Historian CSV-Datei

Abbildung 2: Abbildungen der Programme "Web Historian" und "Historian"

- **Historian** [9]
  Dieses Programm wird als Freeware vertrieben und unterstützt ebenfalls mehrere Browser. Entwickelt wird die Anwendung in Österreich und ist nur für Windows erhältlich. Genutzt werden können die Browser Chrome, InternetExplorer, Firefox und Opera in allen Versionen. Die Bedienung kann Grafisch oder über die Befehlszeile erfolgen. Exportiert werden können die Daten als Text oder CSV-Datei, wobei sich das jeweilige Aussehen über Template-Dateien anpassen lässt. Ein händisch eingebundenes Image kann durch die direkte Übergabe der jeweiligen History-Dateien verarbeitet werden. Eine Automatisierung könnte mittels Skript möglicherweise realisiert werden. In der vorherigen rechten Abbildung ist eine von "Historian" erstellte CSV-Datei mit Hilfe von OpenOffice-Calc geöffnet.

- **Forensik Toolkit** [10]
  Dieses Toolkit, welches abgekürzt als FTK bezeichnet wird, ermöglicht eine Vielzahl an forensischen Untersuchungen für Windows oder OsX, darunter auch das Visualisieren des Browser-Cache. Angezeigt werden können alle Cacheverzeichnisse, bei denen die Daten unverändert abgelegt werden. Außerdem können mit dieser Anwendung SQLite-Datenbanken betrachtet und analysiert werden. Weitere Funktionen in Bezug auf die Browserforensik sind momentan nicht eingebaut.

- **The Sleuth Kit** [11]
  Bei diesem OpenSource-Tool handelt es sich um eine umfangreiche Sammlung von Forensik Programmen für die Befehlszeile mit der Ausrichtung auf Dateisysteme. Es existieren Versionen für alle unixartigen Betriebssysteme. Unter Windows kann die Sammlung durch die Unix-Umgebung Cygwin [12] genutzt werden. Zur grafischen Darstellung der gewonnen Informationen kann das Programm Autopsy, welches vom selben Hersteller verfügbar ist, genutzt werden. Dieses Toolkit kann für das Auffinden von gelöschten oder versteckten Browser-History-Dateien verwendet werden (Alternate Data Streams bei NTFS).

- **PhotoRec** [13]
  Bei diesem OpenSource-Programm handelt es sich um eine Anwendung zum Wiederherstellen von gelöschten Dateien. Die erste Version dieser Anwendung konnte nur Bilddateien wiederherstellen, woraus sich auch der Name abgeleitet hat. Mittlerweile können über 320 verschiedene Dateitypen wiederhergestellt werden. Die Suche nach Dateien kann auch unabhängig vom jeweiligen Dateisystem durchgeführt werden, solange dieses nicht verschlüsselt ist. Die zu suchende Dateitypen können bereits beim Programmstart ausgewählt werden. Bedient wird das Programm von der Befehlszeile aus und ist für Windows und Linux erhältlich. Die nachfolgende Abbildung zeigt “photorec” bei der Suche nach allen möglichen Dateien in einer Linuxpartition.

- **Chrome Cache View** [14]
  Mit Hilfe dieses Freeware-Programms, das nur für Windows erhältlich ist, kann der aktuelle Cache des Browsers Chrome betrachtet werden. Mit Hilfe der Exportfunktion können die einzelnen Dateien in einen definierbaren Ordner kopiert werden.

Abbildung 3: Die Ausgabe von “photorec” während der Bearbeitung

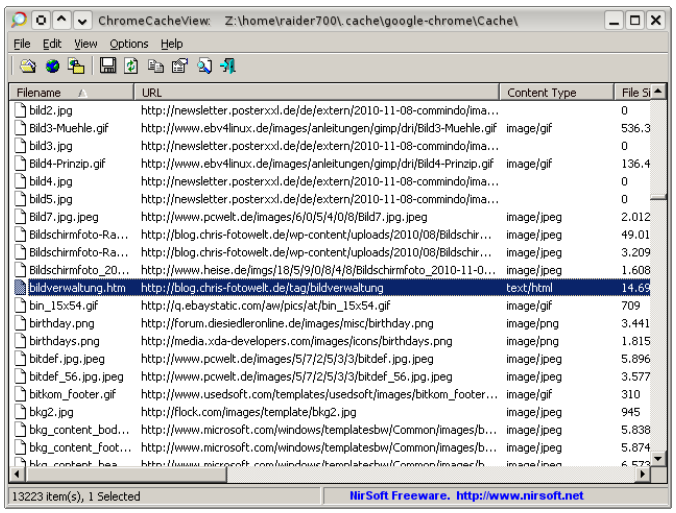


Abbildung 4: Die Oberfläche von “ChromeCacheView” bei geöffnetem Cache.

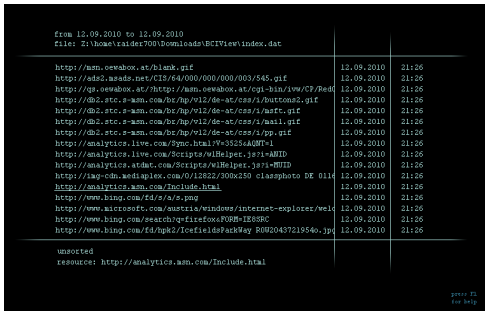
- **Browser Cache Index Viewer** [15]
  Diese Freeware-Anwendung kann genutzt werden, um die “index.dat”, welche die Cacheinformationen enthält, auszulesen. Die gewonnenen Informationen können als XML-Datei exportiert werden. Das Programm benötigt zwar eine grafische Oberfläche, wird jedoch wie ein Befehlszeilenprogramm bedient. Heruntergeladen werden kann es nur für Windows. Abbildung 5(a) auf Seite 15 zeigt die Oberfläche dieses Programms an.

- **Index.dat Zapper** [16]
  Mit Hilfe dieser grafischen Freeware-Anwendung können "index.dat"-Dateien gesucht und die Anzahl der Einträge ausgewertet werden. Bei der kostenpflichtigen Version können die gefundenen Dateien durch mehrfaches Überschreiben dauerhaft gelöscht werden. Zum Anzeigen des Inhaltes wird ein zusätzliches Programm, wie etwa das als nächstes beschriebene, benötigt. In Abbildung 5(b) auf der nächsten Seite wird die Oberfläche dieser Anwendung dargestellt.

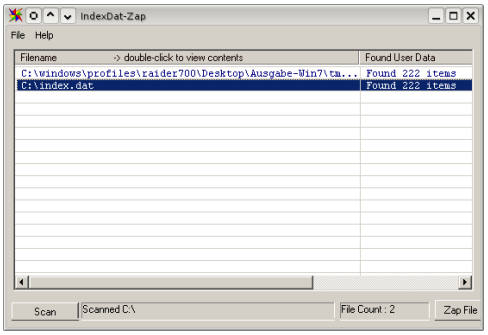
- **Index.dat Viewer** [17]
  Diese grafische Freeware-Anwendung dient dazu, den Inhalt einer "index.dat"-Datei anzuzeigen. Diese Dateien können allerdings nicht über die Anwendungen gesucht werden, wodurch eine Kombination mit dem zuvor erwähnten Programm sinnvoll ist. Die ausgelesenen Daten können als Textdatei exportiert werden. Die grafische Oberfläche dieses Programmes wird in Abbildung 5(c) auf der nächsten Seite dargestellt.

- **IE History Manager** [18]
  Dieses Freeware-Programm für Windows kann ebenfalls "index.dat"-Dateien anzeigen. Zusätzlich können einzelne Einträge entfernt oder geändert werden. Neben den reinen URLs können auch Cookies verarbeitet werden. Eine Exportfunktion existiert ebenso wenig wie eine Suchfunktion. In der nachfolgenden Abbildung 5(d) auf der nächsten Seite ist die Oberfläche der Anwendung abgebildet.

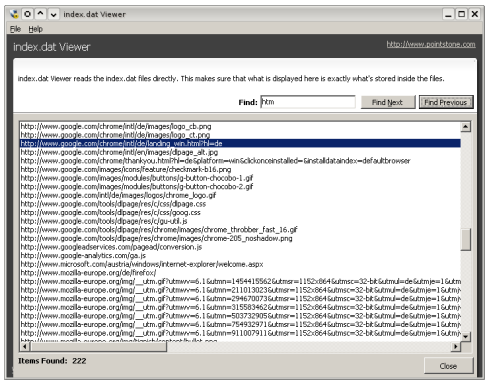
- **BackTrack** [19]
  Dabei handelt es sich weder um eine Anwendung noch um ein Toolkit, sondern um eine Linux Live-CD auf der Basis von Slackware mit der Ausrichtung auf Sicherheit, Passwortwiederherstellung und Forensik. Die angesprochenen Anwendungen pasco, galleta und photorec sind ebenfalls vorhanden. Für forensische Untersuchungen gibt es einen eigenen Eintrag im Bootmenü, wodurch kein Laufwerk automatisch eingebunden und keine SWAP-Partition genutzt wird. Diese Distribution kann auf einer Festplatte oder einem USB-Stick installiert werden. Der Download beträgt zwei Gigabyte, wodurch ein DVD-Rohling zum Brennen benötigt wird. Die nachfolgenden Abbildungen zeigen Backtrack mit zwei der möglichen grafischen Oberflächen.



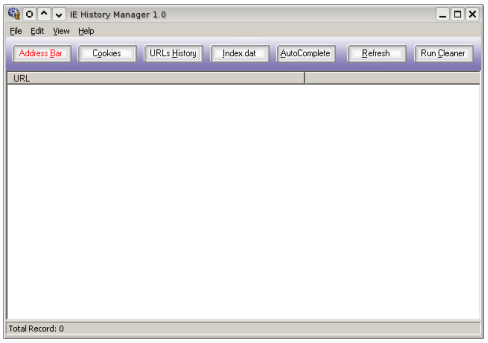
(a) Programmfenster von “Browser Cache Index Viewer”



(b) Programmfenster von “Index.dat Zapper”

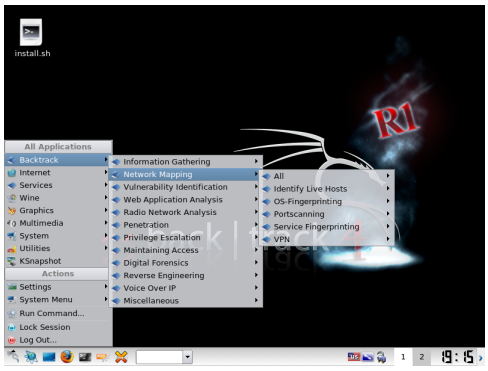


(c) Programmfenster von “Index.dat Viewer”



(d) Programmfenster von “IE History Manager”

Abbildung 5: Abbildungen der Programme “Web Historian” und “Historian”



(a) BackTrack mit KDE

(b) BackTrack mit Fluxbox

Abbildung 6: Grafische Oberfläche Fluxbox und KDE von BackTrack

- **Knoppix-SDT** [20]
  Hierbei handelt es sich ebenfalls um eine Linux Live-CD auf der Basis von Knoppix mit der Ausrichtung auf Security, Passwortwiederherstellung und Forensik. Wie bei BackTrack sind auch hier pasco, galleta und phototrec vorhanden. Auf der Festplatte kann diese Live-CD nicht installiert werden, dafür wird beim Systemstart automatisch eine eventuell vorhandene SWAP-Partition ignoriert und keine Festplatte automatisch eingebunden. Die Größe des ISOs beträgt fast 500 Megabyte und kann somit auf eine normale CD gebrannt werden.

- **Inside Security Rescue Toolkit** [21]
  Dabei handelt es sich ebenfalls um eine Linux Live-CD, auf der Basis von Debian für den Sicherheits- und Forensikbereich. Im Gegensatz zu den beiden bereits erwähnten Live-CDs sind "pasco" und "galleta" nicht installiert Doch dafür benötigt das gesamte ISO weniger als 60 Megabyte und kann somit auf eine Scheckkarten-CD gebrannt werden.

- **Helix** [22]
  Diese Live-CD existiert in zwei Versionen. Die Pro-Version ist kostenpflichtig und bietet mehr Möglichkeiten als die normale aber kostenlose Version an. Heruntergeladen werden kann das 700 Megabyte große Image erst nach einer kostenlosen Registrierung. Helix ist stark auf die forensische Untersuchung von Speichermedien spezialisiert und liefert dementsprechend viele Anwendungen mit. Installiert ist unter anderem das bereits erwähnte Sleuthkit. Die Programme "pasco" und "galleta" sind hingegen nicht installiert.

Neben den angeführten Beispielen gibt es noch viele weitere Programme, welche aber aufgrund des Umfanges dieses Dokuments nicht alle vorgestellt werden können.

4 Browserforensic

Nachdem im vorherigen Kapitel 3 auf Seite 4 der aktuelle Stand der Forschung inklusive alternativer Anwendungen besprochen wurde, wird in diesem Kapitel genauer auf die einzelnen Browser eingegangen. Zunächst werden in Unterkapitel 4.1 allgemeine Punkte zu den Browsern behandelt. Im Anschluss daran werden die vier Browser genauer beschrieben.

4.1 Grundlagen

Bei der Browserforensic müssen ebenso wie bei anderen computerforensischen Untersuchungen die beiden folgenden Punkte zwingend eingehalten werden [23]:

- **Authentizität:**
  Es muss sichergestellt werden, dass die zu bearbeitenden Daten mit den originalen vollständig übereinstimmen. Ein gültiger Beweis dafür sind Prüfsummen (checksum) oder Hashwerte, wie sie beispielsweise von MD5 oder SHA erzeugt werden. Stimmt der Hashwert der originalen Datei mit der zu bearbeitenden überein, so besitzen beide denselben Inhalt. Es existieren zwar zu jedem Hashwert unzählige weitere Dateien, aus denen dieser ebenfalls gebildet werden kann. Allerdings ist die Wahrscheinlichkeit, dass eine dieser Dateien einen für den Bereich sinnvollen Inhalt besitzt, nahezu null.

- **Integrität:**
  Während der kompletten Bearbeitung muss jeder Schritt von einer vertrauenswürdigen Person dokumentiert werden, um einen lückenlosen Nachweis der Bearbeitung zu gewährleisten. Diese vollständige Überwachung und Dokumentation wird als “chain of custody” bezeichnet. Durch jede noch so kleine nicht dokumentierte Stelle kann das endgültige Ergebnis angefochten werden, wodurch die Untersuchung wiederholt werden müsste. Für teil oder vollautomatisierte Programme kann der Nachweis über den Quellcode und die reproduzierbare Ausführung erfolgen.

Für die Browserforensic bedeutet dies, dass das komplette System so wenig wie möglich verändert werden darf. Der Idealfall ist, dass überhaupt keine Änderung stattfindet. Dazu bietet es sich an, die Untersuchung unabhängig vom jeweiligen Betriebssystem durchzuführen, indem ein Image der Festplatte erstellt und dieses auf einem anderen

Rechner verarbeitet wird. Dabei muss aber auch nachgewiesen werden, dass zu keinem Zeitpunkt ein schreibender Zugriff auf die Datei erfolgte. Alternativ zum Image kann auch die komplette Festplatte über einen Schreibblocker an ein anderes System angehängt werden. Steht kein solcher Hardwareschutz zur Verfügung, muss dies über die Software sichergestellt werden, was gerade unter Microsoft Windows ein Problem darstellt.

Der Nachweis der Integrität ist hingegen leichter, allerdings abhängig vom genutzten Auswertungsprogramm. Ergibt sich durch mehrmaliges Ausführen immer die identische Ausgabedatei, so ist die Reproduzierbarkeit gewährleistet. Liegt dann noch der Quellcode des Programms vor, kann überprüft werden, dass keine Daten auf irgendeine Weise manipuliert wurden. Damit ist auch die komplette Integrität gewährleistet. Liegt kein Quellcode vor, so muss der Hersteller diesen Nachweis vollbringen, da sonst die Anwendung nicht genutzt werden sollte.

Bei der Auswertung ergibt sich oft ein Problem mit binären Daten. Diese sind oft unzureichend dokumentiert, sodass die Verarbeitung stark erschwert oder komplett verhindert wird. Dies trifft beispielsweise auf den Browser Opera zu, welcher in Kapitel 4.3 auf Seite 23 beschrieben wird. Bei diesem konnten die Cookies genau aus diesem Grund nicht ausgewertet werden.

Wie bereits erwähnt, kann unter Microsoft Windows ein Schreibschutz auf eine Festplatte oder ein Image ohne tiefgehende Systemveränderungen nur mit einem Hardwareschutz in Form eines Readonly-Adapters garantiert werden. Deshalb empfiehlt es sich, als Basis für eine Untersuchung ein auf Unix basierendes System zu nutzen, da dort die systeminternen Vorgänge einfacher kontrolliert und damit auch ein Schreibschutz direkt beim Einbinden sichergestellt werden. Ein zusätzlicher Einsatz eines Hardwareschutzes bietet einen redundanten Schutz und damit eine solide Basis für weitere Arbeiten.

4.2 Firefox

Der Browser Firefox wird von der Mozilla Foundation entwickelt [24]. Die zum Zeitpunkt der Erstellung dieser Arbeit aktuelle Version trägt die Nummer 3.6.12 und kann von der Mozilla-Webseite heruntergeladen werden. Der Browser wurde in C++ geschrieben und nutzt das GTK-Framework für die grafische Ausgabe. Zum Darstellen der Webseiten

wird die Gecko-Engine genutzt. Verwendet werden kann der Browser unter Windows, Linux und OS X werden. Außerdem wird er unter mehreren Lizenzen vertrieben (MPL, GPL und LGPL).

Bis Version 3 hat Firefox eine binäre Datei für die Historydaten benutzt. Seit Version 3.0 wird hingegen eine SQLite-Datenbank verwendet. Die Inhalte können so durch eine SQL-Abfragen ausgelesen werden. Die genauen Möglichkeiten werden in Kapitel 4.2.2 auf der nächsten Seite beschrieben. In der im Zuge dieser Arbeit erstellten Anwendung werden nur die Versionen ab 3.0 unterstützt, weil alle Versionen bis 3.0 weniger als 1% aller Firefoxinstallationen ausmachen und die Zahl kontinuierlich weiter abnimmt [25]. Zur kommenden Version 4 ändert sich, zumindest bei Tests mit der ersten Betaversion, nichts an der Struktur des Cache. Dadurch ist auch voraussichtlich keine Anpassung der Anwendung bei der finalen Version notwendig.

4.2.1 Genutzte Dateien

Das Benutzerverzeichnis des Browsers befindet sich abhängig vom jeweiligen Typ des Betriebssystems an folgender Stelle:

- **Windows Vista/7:**
  Der Ordner ist im Verzeichnis des Benutzers unter *"AppData/Roaming/Mozilla/Firefox/Profiles/"* zu finden. Der zugehörige Cachepfad befindet sich unter *"AppData/Local/Mozilla/Firefox/Profiles/"* und muss daher eigens berücksichtigt werden.

- **Windows 2000/XP:**
  Im Verzeichnis des Benutzers ist der Ordner unter *"Anwendungsdaten/Mozilla/Firefox/Profile"* zu finden. Der zugehörige Cacheordner liegt unter *"Lokale Einstellungen/Anwendungsdaten/Mozilla/Firefox/Profile"* und muss ebenfalls berücksichtigt werden.

- **Linux:**
  Unter Linux ist das Verzeichnis, das im Ordner *".mozilla/firefox"* liegt, im Standard-Verzeichnis des Benutzers zu finden. Der Cacheordner ist ebenfalls dort als Unterordner anzutreffen.

Das Programm nutzt beide unterschiedlichen Verzeichnispositionen für die Auswertung. Genutzt werden folgende im Browserverzeichnis vorhandene Dateien:

- **places.sqlite**
  In dieser SQLite-Datei werden sämtliche URL-Informationen gespeichert. Die genutzten auslesbaren Daten werden im nächsten Kapitel 4.2.2 behandelt.

- **cookies.sqlite**
  Dabei handelt es sich ebenfalls um eine SQLite-Datei. Dort werden alle informationen zu den Cookies gespeichert.

- **_CACHE_001_ bis _CACHE_003_**
  Bei diesen Dateien handelt es sich um die Cachedateien, die unter anderem auch die Zuordnungen der URLs zu den anderen Dateien im Cacheordner bieten. Bei kleineren HTML-Dateien sind diese auch direkt in der jeweiligen Datei hinterlegt.

- **Alle Dateien im Ordner Cache**
  In diesem Ordner liegen die Originaldateien, die heruntergeladen wurden. Der Name wurde dabei verändert, die Zuordnung ist jedoch in den zuvor erwähnten Dateien zu finden.

Es existieren zwar noch viele weitere Dateien im Browserverzeichnis, diese wurden aber für die Auswertung nicht benötigt. Dabei handelt es sich beispielsweise um allgemeine Plugins, integrierte Suchmaschinen oder um Lesezeichen.

4.2.2 Auslesemöglichkeiten

Zum Auslesen der Informationen wird ein SQLite-Programm benötigt. Für die im Zuge der Arbeit erstellten Applikation wird “sqlite3” genutzt. Folgende Informationen über die URLs können dadurch gewonnen werden:

- **“url”**
  In diesem Feld wird die genaue jeweilige URL gespeichert.

- **“title”**
  Hier wird der zur URL gehörende Webseitentitel hinterlegt.

- **“rev_host”**
  Der invertierte Hostname wird in dieses Feld eingetragen. So wird beispielsweise aus “www.google.at” der String “ta.elgoog.www”.

- **“visit_count”**
  Die Anzahl der Besuche der jeweiligen URL wird hier eingetragen.

- **“hidden”**
  Hier wird angegeben, ob es sich um eine im Hintergrund geladene URL handelt. Ist dies der Fall, würde die URL nie bei den Vorschlägen in der Adresszeile des Browsers aufscheinen.

- **“typed”**
  In diesem Feld wird angegeben, ob die URL vom Benutzer eingegeben wurde.

- **“id”**
  Wenn der “visit_count” größer als null und die URL nicht “hidden” ist, wird eine eindeutige Nummer zur Referenzierung vergeben.

- **“from_visit”**
  Dieses Feld gibt an, von welcher URL aus der Aufruf erfolgt ist. Der Eintrag bezieht sich dabei auf die zuvor erwähnte “id”.

- **“visit_type”**
  Die in diesem Feld eingetragene Zahl gibt einen von sieben nachfolgend aufgelisteten möglichen Besuchstypen an:

Nr.	Bedeutung
1	Die URL wurde durch einen angeklickten Link aufgerufen.
2	Die Adresse wurde aus der History aufgerufen.
3	Der Aufruf der URL erfolgte aus einem Lesezeichen.
4	Die URL wurde als ein in HTML eingebetteter Inhalt aufgerufen.
5	Hierbei handelt es sich um eine permanente Weiterleitung.
6	Hierbei handelt es sich um eine temporäre Weiterleitung.
7	Bei der Adresse handelt es sich um einen Download.

- **“visit_date”**
  Hier wird der Zeitpunkt des ersten Besuches hinterlegt.

- **“last_visit_date”**
  Der Zeitpunkt des letzten Besuches wird in diesem Feld eingetragen.

- **“session”**
  Gibt an, in welchem Browser-Tab die gefundene URL geöffnet wurde.

Die Cookies können ebenfalls mittels SQL-Befehlen ausgelesen werden. Dabei können die nachfolgenden Informationen gewonnen werden:

- **“id”**
  Hier wird eine ID für den jeweiligen Cookie gesetzt. Diese setzt sich aus dem Zeitpunkt der Erstellung zusammen und unterscheidet sich bei zusammengehörenden Cookies nur an der letzten Stelle.

- **“name”**
  In dieses Feld wird der Name des Cookies eingetragen, welcher pro Domain einzigartig sein muss.

- **“value”**
  Die Daten des jeweiligen Cookies werden in dieses Feld eingetragen. Dabei kann es sich um eine einfache ID oder eine komplette Signatur zur Wiedererkennung handeln.

- **“host”**
  Die Zugehörigkeit zu einer Domain wird hier hinterlegt.

- **“path”**
  Gibt den Pfad des Cookies in Bezug auf den Server an.

- **“expiry”**
  In dieses Feld ist der Ablaufzeitpunkt des Cookies eingetragen.

- **“lastAccessed”**
  Hier wird angegeben, wann der letzte Zugriff auf dieses Cookie war.

- **“isSecure”**
  In dieses Feld wird eingetragen, ob es sich um einen verschlüsselten Cookie handelt.

- **“isHttpOnly”**
  Hier wird hinterlegt, ob es sich um einen Cookie für nur HTTPS Verbindungen handelt.

Zu beachten ist, dass alle angegebenen Zeiten in Unix-Time mit 100 Nanosekunden-Intervall angegeben sind. Zur späteren Verarbeitung muss diese Zeit angepasst werden, indem die Zahl durch eine Million dividiert und dadurch in Sekunden dargestellt wird.

4.3 Opera

Der Browser Opera wird von der gleichnamigen Firma entwickelt [26]. Die zum Zeitpunkt der Erstellung dieser Arbeit aktuelle Version ist 10.63 und kann für Windows oder Linux heruntergeladen werden. Programmiert ist der Browser in C++. Für die Ausgabe wird ein von Opera entwickeltes Toolkit genutzt, welches QT und GTK nativ ansteuern kann. Der Browser passt sich somit an die jeweilige Umgebung direkt an. Vertrieben wird das Programm kostenlos unter einer proprietären Lizenz.
Seit Version 6 nutzt Opera eine Datei namens “history.dat” oder “global_history.dat” zum Speichern der besuchten Webseiten. Darin sind allerdings nur wenige Informationen enthalten, worauf in Kapitel 4.3.2 auf der nächsten Seite genauer eingegangen wird.

Dieser Browser bereitet beim automatisierten Auswerten die meisten Probleme, da die Cookies und der Cache in einem binären Format vorliegen und dazu öffentlich keine detaillierten Informationen existieren. Kontaktversuche mit der Firma blieben leider unbeantwortet, wodurch die Cookies gar nicht bearbeitet werden konnten.

4.3.1 Genutzte Dateien

Die Browserverzeichnisse befinden sich je nach Systemtyp für einen Benutzer an folgender Stelle:

- **Windows Vista/7:**
  In den aktuellen Windowsversionen befindet sich der Ordner im Benutzerverzeichnis unter *“AppData/Roaming/Opera/Opera”*, die zugehörigen Cache-Daten unter *“AppData/Local/Opera/Opera”*

- **Windows 2000/XP:**
  Hier sind die Verzeichnisse ebenfalls in zwei Ordner aufgeteilt. Die History-Daten

sind unter *"Anwendungsdaten/Opera/Opera"* und *"Lokale Einstellungen/Anwendungsdaten/Opera/Opera"* im Benutzerverzeichnis angeführt.

- **Linux:**
  Auf diesem System befinden sich alle Daten im einzigen Ordner *".opera"*, welcher im Standard-Benutzerordner liegt.

Wie bereits im vorherigen Kapitel erwähnt, konnten nur wenige Informationen von diesem Browser gewonnen werden. So konnte etwa die Datei "cookies4.dat" überhaupt nicht bearbeitet werden. Für die Auswertung wurden folgende Dateien genutzt:

- **history.dat oder global_history.dat**
  In dieser Textdatei stehen die einzelnen URLs inklusive drei weiterer Optionen untereinander. Genauere Angaben dazu befinden sich im nächsten Kapitel 4.3.2.

- **dcache4.url**
  Bei dieser Datei handelt es sich um die zentrale Cachedatei. Dort sind alle Zuordnungen zu den jeweiligen Cachedateien enthalten.

- **Alle Dateien in den Ordnern "g_*"**
  Der Stern steht hierbei für "0001" bis "000n". In diesen Ordnern liegen die heruntergeladenen Dateien, die in "opr<Nummer>" umbenannt wurden. Der Inhalt der jeweiligen Datei wird allerdings nicht verändert.

Die weiteren Dateien im Browserverzeichnis beinhalten Lesezeichen, eingegebene Suchwörter oder die installierten Plugins. Für die Auswertung wurden diese Dateien allerdings nicht genutzt.

4.3.2 Auslesemöglichkeiten

Bei diesem Browser konnten nur folgende vier Informationen ausgelesen werde:

- **NAME**
  Der Name der zur URL gehörenden Webseite. Existiert dieser nicht, so wird hier ebenfalls die URL eingetragen.

- **URL**
  Hier wird die genaue Adresse der URL hinterlegt.

- **TIME**
  Diese in Unix-Time codierte Zeit gibt den Zeitpunkt des Besuches an.

- **NUMBER**
  Zusätzlich kann eine Nummer ausgelesen werden, welche bis auf wenige Ausnahmen immer "-1" ist. Der Zweck dieser Nummer ist ebenfalls in keiner öffentlich hinterlegten Dokumentation zu finden.

Der Aufbau der Datei selbst ist interessant, da die erwähnten vier Optionen jeweils in eine eigene Zeile gespeichert werden und sich pro Eintrag wiederholen. Da neue Einträge einfach angehängt werden, ergibt sich automatisch eine chronologische Sortierung.

4.4 Chrome

Der Browser Chrome wird von Google entwickelt [27]. Das Programm wird in C++ und Java geschrieben und nutzt ein eigenes Framework zur Anzeige. Der Quellcode steht unter OpenSource-Lizenz von BSD und nennt sich Chromium. Chrome basiert genau auf diesen Sourcen und bietet zusätzlich noch Videocodecs für h.264 an. Der Browser kann unter anderem für Windows und Linux heruntergeladen werden.
Der Aufbau der History-Datei ist ähnlich dem des Firefox. Es wird ebenfalls eine SQLite-Datenbank für die History und eine weitere für die Cookies genutzt. Genaueres dazu wird in den beiden nächsten Unterkapiteln angeführt.

Interessant ist, dass alle verwendeten Zeitpunkte nicht wie sonst üblich auf der "Unix-Time" basieren, sondern auf der "Filetime". Erstere zählt die vergangenen Sekunden seit Mitternacht 1.1.1970, letztere die vergangene Zeit in Nanosekunden seit Mitternacht 1.1.1601. Zur Umrechnung muss der Wert daher durch eine Million dividiert werden, um auf Sekunden zu kommen. Anschließend wird beim Ergebnis noch die Differenz der 369 Jahre, das sind 11.644.473.600 Sekunden, abgezogen.

4.4.1 Genutzte Dateien

Die Verzeichnisse des Browsers sind, gleich wie die vorherigen, abhängig vom Typ des Betriebssystems:

- **Windows Vista/7:**
  Der Ordner befindet sich inklusive Cache im Benutzerverzeichnis unter *"AppData/Local/Google/Chrome/User Data/"*. Eine Unterscheidung zwischen "Local" und "Roaming" gibt es nicht.

- **Windows 2000/XP:**
  Bei älteren Windowsversionen befindet sich der Cache-Ordner ebenfalls direkt im History-Ordner. Dieser ist vom Benutzerverzeichnis aus unter *"Lokale Einstellungen/Anwendungsdaten/Google/Chrome/User Data/"* zu finden.

- **Linux:**
  Als einziger der vier Browser hält sich Chrome komplett an die Linux-Standard-Base (LSB) und nutzt zwei Ordner. Die History-Daten liegen im Benutzerordner unter *".config/google-chrome/"*. Die Cache-Daten sind unter *".cache/google-chrome/"* ebenfalls im Benutzerordner zu finden. Zum Auswerten müssen daher beide Ordner berücksichtigt werden.

Die nachfolgenden Dateien werden für das Auslesen der Daten währen der Verarbeitung benötigt:

- **History**
  Bei dieser Datei ohne Endung handelt es sich um eine SQLite-Datenbank. In dieser werden sämtliche besuchten History-Daten gespeichert. Welche Felder ausgelesen werden können, ist in Kapitel 4.4.2 auf der nächsten Seite angeführt.

- **Cookies**
  Hier handelt es sich ebenfalls um eine SQLite-Datenbank, in der sämtliche Informationen zu den einzelnen Cookies gespeichert werden.

- **data_0 bis data_3**
  Diese Dateien beinhalten die Zuordnung der URLs zu den einzelnen Cache-Dateien.

- **Alle Dateien im Ordner Cache**
  Bei allen Dateien, welche mit "f_" beginnen, handelt es sich um umbenannte originale Dateien. Die zuvor erwähnten "data_*"-Dateien beinhalten die genaue Zuordnung.

Die anderen Dateien im Browser-Benutzerordner beinhalten unter anderem Informationen zu Lesezeichen, Tabs, Vorschaubildern und Logindaten. Für die im Zuge der Arbeit erstellten Anwendung werden diese Daten allerdings nicht genutzt.

4.4.2 Auslesemöglichkeiten

Da es sich bei den History-Dateien um SQLite-Datenbanken handelt, wird ein Programm wie "sqlite3" zum Auslesen benötigt. Folgende Informationen können aus der Datei "History" mittels SQL-Befehl ausgelesen werden:

- **"id"**
  Dieses Feld beinhaltet eine eindeutige Nummer für jeden Eintrag.

- **"from_visit"**
  Hier wird die eindeutige Nummer der vorherigen URL eingetragen.

- **"is_indexed"**
  Dieses Feld gibt an, ob sich die URL im Index der Datenbank befindet.

- **"url"**
  Hier wird die genaue Web-Adresse hinterlegt.

- **"title"**
  Der Titel, der zur URL gehörenden Webseite wird in dieses Feld eingetragen. Sollte dies nicht möglich sein, so bleibt das Feld einfach leer.

- **"visit_count"**
  Durch die laufende Nummer wird angegeben, wie oft diese Adresse aufgerufen wurde.

- **"typed_count"**
  In diesem Feld wird angegeben, wie oft die Adresse händisch eingegeben wurde.

- **"visit_time"**
  Hier ist der Zeitpunkt des ersten Besuches hinterlegt. Als Format wird die Filetime genutzt.

- **“last_visit_time”**
  In dieses Feld wird der Zeitpunkt des letzten Besuches, ebenfalls in Format der Filetime, eingetragen.

- **“hidden”**
  Hier wird angegeben, ob es sich um eine versteckte URL handelt. Ist dies der Fall, so erscheint dieser Eintrag nie bei den automatischen Vorschlägen zur Vervollständigung der Adresse.

Zusätzlich können folgende Informationen aus der Datei “Cookies” gewonnen werden:

- **“host_key”**
  In dieses Feld wird der Name der Domain mit voranstehendem Punkt, aber ohne Sub-Domain wie “www“ eingetragen.

- **“name”**
  Hier wird der Name des Cookies hinterlegt. Pro Domain darf dieser Name nur einmal existieren.

- **“value”**
  An dieser Stelle ist die eigentliche Information des Cookies zu finden. Der Inhalt kann von einer einfachen ID bis zu einer digitalen Signatur alles enthalten.

- **“path”**
  In diesem Feld ist der Pfad des Cookies in Bezug auf den Server hinterlegt.

- **“creation_utc”**
  Hier wird der Erstellungszeitpunkt in 100-Nanosekunden genau angegeben.

- **“last_access_utc”**
  Der letzte Zugriff auf den Cookie wird hier als Zeitstempel hinterlegt.

- **“expires_utc”**
  In das dritte Zeitfeld wird das Ablaufdatum des jeweiligen Cookies hinterlegt.

- **“secure”**
  Dieses Feld gibt an, ob es sich um einen Cookie mit verschlüsselten Werten handelt.

- **“httponly”**
  Hier kann noch ausgelesen werden, ob der Cookie nur für HTTPS-Adressen gilt.

Wie im direkten Vergleich zu Kapitel 4.2.2 auf Seite 20 zu sehen ist, bietet der Browser Chrome viele Ähnlichkeiten zum Firefox. Bei der Implementierung konnten daher auch einige Codestellen übernommen werden.

4.5 Internet Explorer

Der Browser InternetExplorer wird von Microsoft entwickelt und direkt mit dem eigenen Betriebssystem vertrieben [28]. Als Download gibt es nur eine Version für Windows.
Zum Auslesen der binären History-Datei “index.dat” wird ein externes Programm namens “pasco” [7] genutzt. Dieses steht unter einer freien Lizenz und kann die erwähnten Dateien in einen lesbaren Text überführen. Für die Cookies gibt es vom selben Programmierer ein Programm namens “galleta” mit dem selben Zweck. Beide werden bei dem hier dargestellten Programm verwendet.

4.5.1 Genutzte Dateien

Der InternetExplorer nutzt mehrere Verzeichnisse für seine History-Dateien, welche alle den Namen “index.dat” tragen.

- **/Cookies**
  Diese “index.dat” beinhaltet die Namen und Zuordnungen der einzelnen Cookies, welche sich im selben Ordner befinden.

- **/Temporary Internet Files/Content.IE5**
  Hier werden in der “index.dat” sämtliche besuchten Adresse eingetragen. Dabei handelt es sich um die für die Auswertung benötigten primären Daten. Die Cachedateien befinden sich ebenfalls an dieser Position in angegebenen Unterordnern.

- **/History/History.IE5**
  In dieser “index.dat” wird hinterlegt, welche Zeiträume der Cache für die jeweils vorhandenen Unterordner abdeckt.

- **/Internet Explorer/userData**
  In dieser Datei werden Informationen vom jeweiligen Benutzer hinterlegt.

Die Verzeichnis sind unter Windows 2000 und XP vom Benutzerordner aus unter "Anwendungsdaten" zu finden. Unter Windows Vista und Windows 7 befinden sich diese vom Benutzerverzeichnis aus gesehen im Ordner "AppData/Local/Microsoft/".

4.5.2 Auslesemöglichkeiten

Die Auslesemöglichkeiten sind auf jene Daten beschränkt, die von "pasco" und "galleta" geliefert werden. Bei den History-Daten handelt es sich um folgende Felder:

- **"Type"**
  Bei diesem Feld wird angegeben, ob es sich um eine direkte URL ("URL") eine Weiterleitung ("REDR") oder spezifische Befehle von Microsoft ("LEAK") [29] handelt.

- **"Type_Extra"**
  Hier wird angegeben, um welchen Typ von Eintrag es sich handelt. Möglich sind beispielsweise "http", "https" oder "cookie".

- **"URL"**
  Hier ist die eigentliche Adresse des Eintrages hinterlegt.

- **"Access_Time"**
  In diesem Feld befindet sich der Zeitpunkt des ersten Besuches, welcher von Pasco als Textstring ausgegeben wird.

- **"Modified_Time"**
  Der letzte Zugriff auf diese URL aktualisiert diesen Zeitpunkt, welcher ebenfalls als Textstring erstellt wird.

- **"Filename"**
  Dies gibt den Dateinamen der im Cache vorhandenen zugehörigen Datei an. Mit Hilfe des nachfolgenden Punktes kann die Datei einfach aufgefunden werden.

- **"Directory"**
  Neben dem Dateinamen wird auch der Unterordnername abgelegt. Durch diesen lässt sich der Pfad für die weitere Auswertung anhand des Dateinamens finden.

- **“HTTP_Headers”**
  An Stelle des Titels einer HTML-Seite wird beim InternetExplorer, ebenso wie bei den anderen Browsern, der genaue HTML-Header gespeichert.

Bei den Cookies können nachfolgende Informationen ausgelesen werden:

- **“SITE”**
  In diesem Feld wird der Domainname ohne Sub-Domain aber mit abschließenden “/“ gespeichert.
- **“VARIABLE”**
  Hier wird der Name des Cookies, welcher pro Domain einzigartig ist, eingetragen.
- **“VALUE”**
  In diesem Bereich sind die eigentlichen Informationen des Cookies hinterlegt. Dabei kann es sich beispielsweise um eine einfache ID oder um eine komplette digitale Signatur handeln.
- **“CREATION_TIME”**
  Der Erstellungszeitpunkt wird von “galleta” als Textstring ausgegeben.
- **“EXPIRE_TIME”**
  Der Ablaufzeitpunkt des Cookies wird ebenfalls als Textstring ausgegeben.

Alle Zeitpunkte wurden bei diesem Programm direkt in die UNIX-Time konvertiert. Als Hilfsmittel dient die Anwendung “date”, die definierte Strings verarbeiten kann.

5 Zielsetzung

Das Ziel der Masterarbeit bestand darin, ein Programm zu erstellen, das aus einem übergebenen Festplatten-Image herausfindet, wann jeder dort vorhandene Benutzer mit einem der unterstützten Browser eine Webseite besucht hat und, ob die gefundenen Daten mit den der über die jeweilige Webadresse abrufbaren Daten übereinstimmen.

Folgende Punkte wurden dabei als Vorgabe definiert:

- **URL:**
  Es sollten alle eingegebenen und besuchten URLs gefunden werden.
- **History:**
  Die Daten sollten aus der History der jeweiligen Web-Browser direkt extrahiert werden.
- **Cache:**
  Der zugehörige Cache sollte beim Validitätscheck mitberücksichtigt werden.
- **Cookies:**
  Die auf der Festplatte gefundenen Cookies sollten der jeweiligen Seite zugeordnet und ebenfalls geprüft werden.
- **Zeitlinie:**
  Eine Zeitlinie für gefundene Daten sollte berücksichtigt werden. Gewonnene Daten sind somit im zeitlich korrekten Ablauf auszugeben.
- **Sessions:**
  Die besuchten Zeiten sollten zu Sitzungen zusammengefasst werden.
- **Überprüfung:**
  Die gefundenen Daten sollten mit den Onlinedaten verglichen und auf deren Korrelation hin getestet werden.
- **Ausgabe:**
  Die Ausgabe der aufbereiteten Daten sollte als XML-Datei erfolgen, damit diese einfach in eine Webseite integriert und damit visualisiert werden kann.

- **Textbasiert:**
  Das Programm sollte auf der Konsole gestartet werden und in dieser auch ohne grafische Oberfläche laufen.

Zusätzlich zu den vorgegebenen Anforderungen wurden folgende Ziele gesetzt:

- **Mehrere Startmöglichkeiten:**
  Es sollte ausgewählt werden können, ob das Programm vollständig automatisiert durchläuft oder interaktive Möglichkeiten bietet.

- **Konfigurierbarkeit:**
  Anpassungen des Programms, welche den Ablauf oder die Pfade betreffen, sollten einfach durchführbar sein.

- **Auswahlsmöglichkeiten:**
  Schon vor dem Start sollte es mittels Startparameter möglich sein, gewisse Punkte wie gewünschte Benutzernamen, gewünschte Browser und Testmöglichkeiten einzuschränken.

- **Gelöschte Dateien:**
  Das Image sollte auf gelöschte Browser-History-Dateien hin durchsucht werden können. Gefundene Dateien werden anschließend bei der Bearbeitung mitberücksichtigt.

- **XML-Ausgabe:**
  Die geplante XML-Ausgabe sollte auf mehrere Arten erfolgen können. Dadurch kann die spätere Visualisierung vereinfacht werden.

- **Festplatten:**
  Neben einem Image einer Festplatte sollte auch die eigentliche Festplatte direkt nutzbar sein.

- **Live-CD:**
  Das Programm sollte ebenso in eine Linux Live-CD integriert werden, damit auch Rechner ohne das Ausbauen der Festplatte und ohne Manipulation der Daten getestet werden können.

Inwieweit jedes der vorgegebenen und zusätzlichen Ziele erfüllt werden konnten, und wodurch es zu möglichen Problemen kommen kann, ist in Kapitel 7.8 auf Seite 122 angeführt.

Im nachfolgenden Kapitel wird das der Zielsetzung entsprechende Programm detailliert besprochen. Außerdem wird neben der Beschreibung auch auf die einzelnen Quellcode-Dateien eingegangen.

6 Programmaufbau

In diesem Kapitel wird das im Zuge dieser Arbeit erstellte Programm ausführlich besprochen. Der volle Name der Anwendung lautet “Web-History-Forensik-Tool”, welches im nachfolgenden auch oft als “WHFT“ abgekürzt wird.

Zunächst wird im ersten Unterkapitel 6.1 auf den gewählten Lösungsweg eingegangen und mit Alternativen verglichen. Im darauffolgenden Kapitel 6.2 auf Seite 37 werden Grundlagen, Möglichkeiten und Besonderheiten von Bash-Skripten behandelt.

Anschließend wird in Kapitel 6.3 auf Seite 39 der detaillierte fertige Aufbau des erstellen Programmes erklärt. In diesem Kapitel wird außerdem auf den Programmordner und den Ordner für die temporären Dateien während eines Bearbeitungsprozesses eingegangen.

Im Unterkapitel 6.4 auf Seite 50 wird die Möglichkeit zur Sprachänderung beschrieben. Im darauffolgenden Kapitel 6.5 auf Seite 52 werden die für einen erfolgreichen Programmablauf notwendigen Programme aufgelistet und besprochen.

Der eigentliche Ablauf des Programmes wird im Unterkapitel 6.6 auf Seite 56 dargestellt und erklärt. Im Anschluss daran werden im Kapitel 6.7 auf Seite 60 alle einzelnen Skripte auf deren Funktion hin detailliert beschrieben.

6.1 Lösungsansatz

Am Beginn der Arbeit wurde nach einem praktikablen Lösungsweg für die in Kapitel 5 auf Seite 32 besprochene Zielsetzung gesucht. Problematisch war dabei vor allem das Einbinden des Images und das Durchsuchen der darin enthaltenen Dateien und Ordner. Dies legte die Verwendung einer Skriptsprache und dem damit verbundenen besseren Ansteuern von externen Programmen nahe.

Es wurde daher geplant, das Programm als eine Mischung aus Bash und Java für Linux zu implementieren. Im Laufe der Programmierung stellt sich jedoch heraus, dass sämtliche Probleme auch direkt in Bash gelöst werden konnten. Daher konnte auf Java komplett verzichtet werden, was einige Vorteile brachte. In der folgenden Auflistung sind einige davon aufgelistet:

- **Quelle einbinden:**
  Über die Bash und den “mount”-Befehl können Partition-Images oder Device-Images mit allen vom System unterstützten Dateisystemen direkt eingebunden werden. Dasselbe gilt auch für alle erkannten Festplatten. Genaueres dazu ist in Kapitel 6.7.7 auf Seite 67 beschrieben.

- **Ausgabelog:**
  Mit Hilfe des Befehls “tee” konnte ein problemloses Aufzeichnen der gesamten Ausgabe ermöglicht werden. Wie dies funktioniert, wird in Kapitel 6.7.1 erklärt.

- **Dateibearbeitung:**
  Mit Unterstützung der GNU-Tools wie “find”, “awk”, “grep”, “cut”, “sed” oder “ls” und deren Verkettungsmöglichkeiten mittels “|” konnten Dateien und Verzeichnisse einfacher bearbeitet werden. Genauere Informationen zur Programmierung durch Bash sind im nachfolgenden Kapitel 6.2 auf der nächsten Seite zu finden.

- **Stringbearbeitung:**
  Von externen Programmen zurückgegebene Daten sind sehr oft Text-Strings. Dies gilt etwa für die Befehle “find” oder “ls”. Diese können über die Bash (Kapitel 6.7.11) relativ einfach geändert oder weiterverarbeitet werden.

- **Mehrsprachigkeit:**
  Dadurch, dass nur eine Programmiersprache verwendet wurde, war es möglich, eine einheitliche Form zu implementieren und dadurch eine mehrsprachige Ausgabe zu schaffen. Wie dies funktioniert, ist in Kapitel 6.4 auf Seite 50 dargestellt.

Alle erwähnten Punkte hätten auch in Java implementiert werden können. Dies hätte jedoch einen erheblichen Mehraufwand bedeutet, um eine Kompatibilität mit Bash zu ermöglichen.

Das Programm wurde, wie bereits am Anfang dieses Kapitels erwähnt, für Linux erstellt. Der Grundgedanke dahinter war, dass dadurch der Schreibschutz beim Image oder Massendatenträger sichergestellt werden sollte. Zusätzlich können weitere Dateisysteme unterstützt werden. Dazu gibt es auch die Möglichkeit eine Live-CD zu erstellen, auf welcher sich das fertige Programm befindet. Somit können auch Rechner untersucht werden, ohne vorher die Festplatte auszubauen.

Es ist allerdings auch möglich, die Anwendung auf anderen Betriebssystemen auszuführen. Die Voraussetzung dafür ist, dass alle in Kapitel 6.5 auf Seite 52 erwähnten Programme installiert sind und dass die Bash selbst ebenfalls verfügbar ist. Dies trifft im Grunde auf die meisten Unix-Systeme zu. Erfolgreich getestet wurde dies unter anderem mit den Systemen "FreeBSD" [30] und "OpenSolaris" [31] (neuer Name: "Nexenta" [32]).

Eine Ausführung unter Microsoft Windows ist mit Hilfe der "cygwin"-Umgebung [12] ebenfalls möglich. Allerdings ist dort ein Schreibschutz für das Image oder die Festplatte nicht sichergestellt, weil weiterhin die Windows-Bibliotheken angesprochen werden. Zusätzlich würde auch nur NTFS und FAT als Dateisystem unterstützt werden. Es wird daher von der Verwendung unter Microsoft Windows abgeraten.

6.2 Bash Programmierung

Bei der Bash handelt es sich um eine IEEE POSIX kompatible Unix-Shell, welche bei den meisten aktuellen GNU|Linux Distributionen als Standard eingestellt ist und direkt mit den GNU-Tools vertrieben wird. [33]

Diese eignet sich sehr gut zum Erstellen von Skripten, da sie umfangreiche Funktionen bietet und direkt mit dem System interagieren kann, aber dennoch einen übersichtlichen Programmcode ermöglicht. Erstellte Skripte lassen sich auf jedem System mit installierter Bash direkt ausführen. Probleme gibt es dabei nur, wenn die für die verwendeten Befehle notwendigen Programme nicht installiert sind.

Bei entsprechender Programmierung können Shell-Skripte sehr robust sein, indem einzelne Bereiche in weitere Skripte ausgelagert werden. Diese können selbstverständlich auch mehrfach genutzt werden, was eine gute Wiederverwendbarkeit des Codes ermöglicht. Das im Zuge der Arbeit erstellte Programm nutzt genau diese Funktionen, weshalb es aus 42 einzelnen Skripten besteht. Auf diese wird in Kapitel 6.7 auf Seite 60 genauer eingegangen.

Durch diese zusätzlichen Skripte gibt es außerdem zwei zusätzliche erwähnenswerte Funktionen:

1. **Pseudo-Threads:**
   Durch das Anhängen eines "&" an den Aufrufbefehl wird das Programm in einer

neuen Konsole im Hintergrund ausgeführt. Es können daher mehrere Skripte parallel aufgerufen werden. Sollen diese miteinander interagieren, müssen Schnittstellen definiert und implementiert werden. Die Bash selber bietet dazu keine Möglichkeit. In der vorliegenden Arbeit wurde diese Möglichkeit allerdings nicht genutzt.

2. **Absturzsicherheit:**
   Kommt es zu einem kritischen Fehler in einem aufgerufenen Skript, so wird dieses beendet. Das Skript, welches das abgestürzte aufgerufen hat, funktioniert jedoch weiterhin, wodurch Fehler erkannt und behoben werden können. Genau diese Funktion wird zum Beispiel für das Skript "generate-work-db-ie-pasco" verwendet. Kommt es zu einem Fehler in dem externen Programm "pasco" (siehe Kapitel 6.5 auf Seite 52), wird nur die aktuelle "index.dat" nicht vollständig gelesen, das restliche Programm läuft allerdings problemlos weiter.

Die Bash bietet alle wichtigen Basisfunktionen einer modernen Skriptsprache. Aufgrund des Umfanges wird hier nicht näher auf diese Funktionen eingegangen. Stattdessen werden im nachfolgenden Text nur jene Bash-spezifische Punkte kurz besprochen, die auch im erstellten Programm oft genutzt wurden. Dies soll dazu beitragen, den Quellcode besser zu verstehen.

Eine wichtige Funktion ist das "Pipen" und Umleiten von Ein- und Ausgabedaten. Dadurch kann die Ausgabe eines Programms direkt als Eingabe für ein weiteres Programm genutzt werden. Soll eine Datei ausgelesen, deren Inhalt nach Zeilen mit einem bestimmten String durchsucht und das Ergebnis in eine neue Datei geschrieben werden, funktioniert dies folgendermaßen:

```
1 cut beispielsdatei.txt | grep "testwort > gefiltertedatei.txt"
```

Durch den Befehl "cut" wird ein Programm aufgerufen, welches den Inhalt der Datei "beispielsdatei.txt", die sich im aktuellen Ordner befindet, auf der Konsole ausgibt. Durch das Zeichen "|" wird die Ausgabe allerdings an den nachfolgenden Befehl "grep" weitergeleitet. Dieser durchsucht jede Zeile nach dem gesuchten Wort "testwort" und würde dies auf der Konsole ausgeben. Da aber das Zeichen ">" nachgestellt ist, wird die Ausgabe allerdings in die Datei "gefiltertedatei.txt" umgeleitet, welche im selben Ordner erstellt wird. Bei der Weiterleitung in eine Datei gibt es außerdem die Möglichkeit

eine vorhandene Datei, wie im vorherigen Beispiel aufgeführt, zu ersetzen oder an den bestehenden Inhalt mittels “»” anzuhängen.

Die Bash bietet außerdem die Möglichkeit, Befehle in einem String auszuführen und die Rückgabe automatisiert in den String einzubauen. Soll etwa der aktuelle Pfad in einer Variablen gespeichert werden, funktioniert dies folgendermaßen:

```
1 aktuellerpfad="‘pwd‘"
```

Der Variablen “aktuellerpfad” wird ein String zugewiesen. Dies erkennt man an den Anführungszeichen nach dem Gleichheitszeichen. Durch die beiden ‘-Zeichen wird der Unterbefehl “pwd” umschlossen, welcher zuerst ausgeführt wird und den aktuellen Pfad zurück gibt. Dieser wird anschließend als String in der Variablen gespeichert.

Eine weitere Funktion ist das Übergeben von Werten an ein aufzurufendes Skript. Sämtliche dem Befehl zu übergebende Werte werden einfach an den Aufruf angehängt. Als Trennzeichen dient dabei das Leerzeichen. Möchte man einen String mit Leerzeichen übergeben, muss dieser mit Anführungszeichen umgeben werden. Das aufgerufene Skript kann auf dessen Werte einfach mit den Befehlen “$1“, “$2“, ... “$n“ zugreifen. Mit “$#” kann die Anzahl der zu übergebenden Felder ausgelesen werden. Damit sämtliche übergebenen Werte auf einmal ausgegeben werden, kann “$*” genutzt werden.

Um den Umfang des aktuellen Kapitels nicht weiter zu erhöhen, wird auf ganz spezielle Möglichkeiten nicht weiter eingegangen. Besteht ein Interesse an der Bash-Programmierung, so gibt es dazu im Internet zahlreiche gute Tutorials (z.B.: [34]).

6.3 Programmstruktur

Das im Zuge der Arbeit erstellte Programm besteht aus 42 einzelnen Bash-Skripten und einigen abhängigen Daten für Einstellungen und Sprachen. Abbildung 7 auf der nächsten Seite zeigt, wie diese Skripte miteinander verbunden sind. In Kapitel 6.6 auf Seite 56 wird neben dem Programmablauf auch beschrieben, wann welches Skript genau genutzt wird.

Das Programm selbst besteht aus mehreren Bereichen:

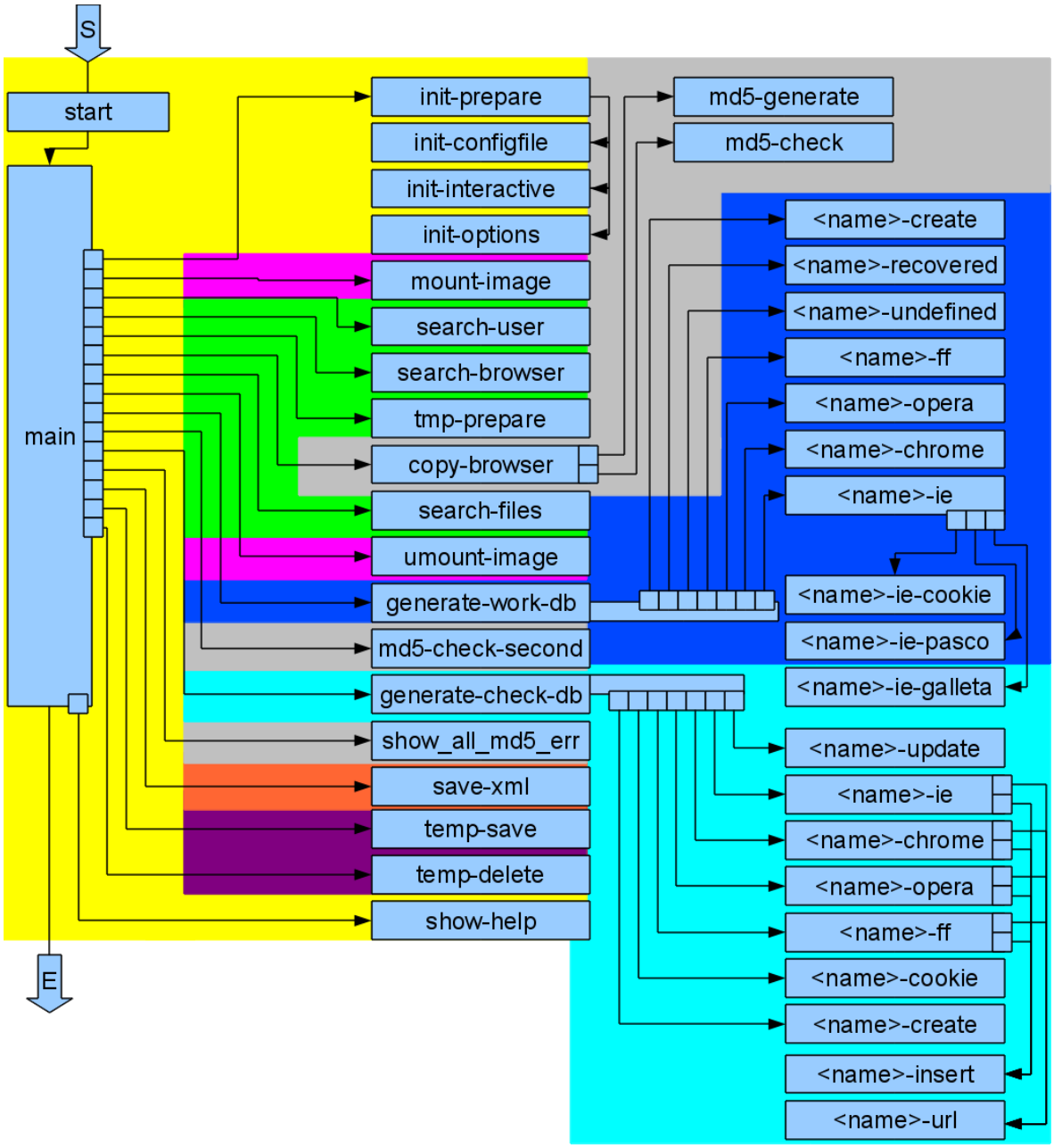


Abbildung 7: Aufbau des erstellten Programms inklusive Teilbereiche

Details zu den einzelnen Skripten sind in den jeweiligen Unterpunkten von Kapitel 6.7 auf Seite 60 zu finden.

- **Initialisierung:**
  In der Abbildung in gelber Farbe dargestellt.
  Dieser Bereich wird beim Programmstart aufgerufen und ist zuständig für das Auswerten der übergebenen Parameter, das Abfragen von Punkten im interaktiven Modus und das Anlegen des temporären Ordners. Ebenso kann die Ausgabe der Programmhilfe diesem Bereich zugeordnet werden. Alle weiteren Bereiche werden ebenfalls von diesem aufgerufen. Das Skript "main" dient dabei als primäres Skript, welches bis zum Programmende durchgehend aktiv ist.

- **Einbinden und Aushängen:**
  Zugehörige Skripte sind mit violetter Farbe hinterlegt.
  Dieser Bereich ist für das Einbinden und Aushängen des Image oder der Festplatte zuständig. Dazu gehört auch die Erkennung der Art der übergebenen Datenquelle, welche ein Partition-Image, ein Disk-Image, eine einzelne Partition oder eine komplette Festplatte sein kann. Der Typ des jeweiligen Dateisystems wird ebenfalls gesucht und - sofern vom System unterstützt - auch eingebunden.

- **Suchen und Wiederherstellen:**
  Dargestellt in der Abbildung in grüner Farbe.
  Dieser ist zuständig für das Suchen des Betriebssystems, des Benutzers und des Browsers im Image. Bei den Betriebssystemen wird unterschieden zwischen "WindowsOLD" (Windows 2000, Windows XP), "WindowsNEW" (Windows Vista, Windows 7) und Linux. Zusätzlich ist er für das Wiederauffinden von gelöschten Dateien nach dem Kopiervorgang zuständig. Dafür wird ein externes Tool namens "photorec" verwendet, welches die Festplatte auf RAW-Ebene durchsucht und Dateien anhand ihrer "Magic number" erkennt und wiederherstellt.

- **Kopieren und Testen:**
  Hinterlegt in der Skizze mit grauer Farbe.
  Ist zuständig für das Kopieren der Dateien vom Image in den temporären Ordner. Dabei wird eine Struktur erstellt, in welcher Betriebssystem, User und Browser zusammengehörig eingefügt werden. Es werden ebenso MD5-Summen von allen

Dateien erstellt, welche immer wieder im späteren Programmverlauf abgefragt werden. Stimmt ein Wert nicht mehr, wird die betreffende Datei in einen speziellen Ordner kopiert und für die weitere Verarbeitung nicht mehr berücksichtigt.

- **Daten auslesen:**
  Bereich wurde mit Dunkelblau hinterlegt.
  Dieser ist zuständig für das Auswerten der gefundenen und kopierten Daten in eine eigene Datenbank. Dabei werden nur mehr die kopierten Ordner aus dem temporären Verzeichnis genutzt. In der Datenbank werden ebenfalls der Betriebssystem-Typ, der Benutzername und der jeweilige Browsername für jeden Eintrag hinzugefügt, um später problemlos den Ursprungsort bestimmen zu können. Wiederhergestellte Dateien werden dabei ebenfalls bearbeitet, jedoch wird als Benutzer- und Systemname "unbekannt" mit einer laufenden Nummer pro Quelldatei eingetragen.

- **Daten testen:**
  Wurde durch eine hellblaue Farbe kenntlich gemacht.
  Dieser Bereich liest die erstellte Datenbank aus, testet dabei die Einträge und fügt die Ergebnisse in eine weitere Datenbank ein. Getestet wird, ob die Seite überhaupt noch existiert, ob der Titel der aktuellen Seite mit der aus der History übereinstimmt, ob ein zugehöriger Cache existiert, ob der existierende Cache identisch mit dem heruntergeladenen ist, wie viele Cookies von der Webseite gesetzt werden, wie viele Cookies im Cache gefunden wurden und wie viele davon identisch sind. Die Ergebnisse dieser Tests werden in die Datenbank eingetragen.

- **Ausgabe erstellen:**
  Markiert durch eine orange Farbe.
  Ist für die Konvertierung der zweiten Datenbank in mehrere definierbare XML-Formate zuständig. Die Unterschiede der einzelnen Arten bestehen in der Zusammenstellung der Hierarchie. So gibt es etwa System-User-Browser oder Browser-System-User. Eine Möglichkeit, jeden Eintrag der Datenbank als gleichwertiges Feld zu exportieren, existiert ebenfalls.

- **Aufräumen:**
  Mit violetter Farbe markiert.
  Dieser Bereich sichert optional den temporären Ordner und löscht diesen anschießend als letzten Schritt von der Programmausführung. Die Sicherung erfolgt als ZIP-Datei in dem Ordner, von welchem das Programm aufgerufen wurde. Ebenso werden die erstellten XML-Dateien dorthin kopiert. Das automatische Loggen der Ausgabe endet mit dem Erstellen des Archivs, da diese Datei im temporären Ordner liegt und damit nicht mehr länger verfügbar ist.

Wie aus den Bereichen erkennbar ist, läuft das Programm relativ linear ab. In welcher Reihenfolge dabei die einzelnen Skripte aufgerufen werden, ist in Kapitel 6.6 auf Seite 56 zu finden.

In den folgenden beiden Unterkapiteln wird der Aufbau auf Dateisystemebene besprochen. Zunächst geht es um den Aufbau des eigentlichen Programms, welcher fix ist und die Skripte mit den notwendigen zusätzlichen Dateien beinhaltet. Anschließend wird der temporäre Ordner behandelt, welcher die Dateien für die jeweils zu bearbeitende Datenquelle enthält.

6.3.1 Programm-Verzeichnis

Das Programmverzeichnis beinhaltet drei Ordner und eine Datei namens “start”. Diese Datei ist für den Start des Programms zuständig und wird vom jeweiligen Anwender aufgerufen. Die Abbildung 8 auf Seite 46 zeigt den Verzeichnisbaum inklusive der darin enthaltenen Dateien an.

Der erste Ordner mit dem Namen “bin” enthält alle einzelnen Skripte, welche in Kapitel 6.7 auf Seite 60 besprochen werden.

Im zweiten Ordner mit dem Namen “etc” ist die Konfigurationsdatei “config.txt” enthalten. Mit dieser lassen sich etwa Standard-Werte, Pfade und Sprache einstellen. Der Inhalt dieser Datei ist folgender:

```
1 # This is the default config for the programme whft
2 # Please be careful with editing one of the lines
3 ################## editable ##################
```

```
4  # dialog art (dialog, Xdialog, gdialog, auto)
5  DIALOG=dialog
6  # languagefile (de,en)
7  LANGFILE=de
8  # session timeout
9  SESSION_TIMEOUT=3600
10 ################## temp path #################
11 # path for temp files
12 TEMPDIR=/tmp/whft
13 # primary tempfile for dialog
14 TEMPFILE=_temp
15 # parameters filename
16 PARAMFILE=params
17 # number of parameters filename
18 PARAMNUMBER=params-number
19 # configurationfile
20 CONFIGFILE=configuration
21 ################## default valuaes #################
22 # browsers
23 BROWSER_OK=firefox,ff,chrome,opera,internetexplorer,ie
24 # define user to use (all oder <name>,<name>, ...)
25 USERNAMES=all
26 # define browser to use (all oder <name>,<name>, ...)
27 BROWSERNAMES=all
28 # search for deleted files (yes, no)
29 SEARCHFILES=no
30 # check found data (yes, no)
31 CHECK=yes
32 # check all cookies (yes, no)
33 COOKIES=yes
34 # define type of output (all, one, single, user, browser, system)
35 OUTPUT=all
36 # save all used data (yes, no)
37 SAVETEMP=yes
38 # interactive mode -> ask after every step (yes, no)
39 INTERACTIVE=no
```

```
40 ################## run ##################
41 # allow the programm to continue (yes, no)
42 RUN=yes
```

Diese Datei wird beim Programmstart ausgelesen und nach der Erstellung in den temporären Ordner ohne Kommentare kopiert. Anschließend werden die übergebenen Parameter auf die temporäre Konfigurationsdatei übertragen. Für nicht übergebene Punkte wird also der eingetragene Wert verwendet. Die jeweils definierbaren Möglichkeiten sind in den Klammern bei den Kommentaren eingetragen. Genauere Informationen zu den einzelnen Punkten sind in Kapitel 7.2 auf Seite 107 zu finden. Der Punkt "RUN" sollte allerdings nicht verändert werden, weil damit die Programmausführung komplett unterbunden wird. Anzumerken ist auch, dass jedes der einzelnen Skripte am Start diese Konfigurationsdatei einliest und nutzt, wodurch die Übergabeparameter zwischen den einzelnen Aufrufen minimiert werden können.

Im dritten Ordner des Programmverzeichnisses mit dem Namen "lang" befinden sich die Sprachdateien. Mit der Fertigstellung des Programms werden als Sprache Deutsch und Englisch unterstützt. Genaueres dazu ist in Kapitel 6.4 auf Seite 50 nachzulesen. Pro Sprache gibt es zwei Dateien, wobei die eine für das eigentliche Programm gedacht ist und die andere nur für die Hilfe. Der Dateiname entspricht dabei dem Kürzel der jeweiligen Sprache und bei der Datei für die Hilfe ist zusätzlich "-help" angehängt.

6.3.2 Temporäres Verzeichnis

Für den Bearbeitungsvorgang existiert ein temporäres Verzeichnis, welches bei jedem Programmstart neu angelegt und mit Daten während der Bearbeitung gefüllt wird. Der Aufbau dieses Verzeichnis ist in Abbildung 9 auf Seite 47 zu sehen. Die in der Skizze fehlenden Ordner sind in Abbildung 10 auf Seite 48 dargestellt. Einträge, welche mit spitzen Klammern umrundet sind, bedeuten eine dynamische Anpassung mit laufender Nummer oder eigenem Namen. Die Anzahl der generierten Verzeichnisse ist dabei nur von der verwendeten Datenquelle abhängig.

Der Zweck der einzelnen Dateien ist folgender:

- Die Datei "bashlog" beinhaltet nach der Programmausführung sämtliche auf der Konsole ausgegebenen Texte.

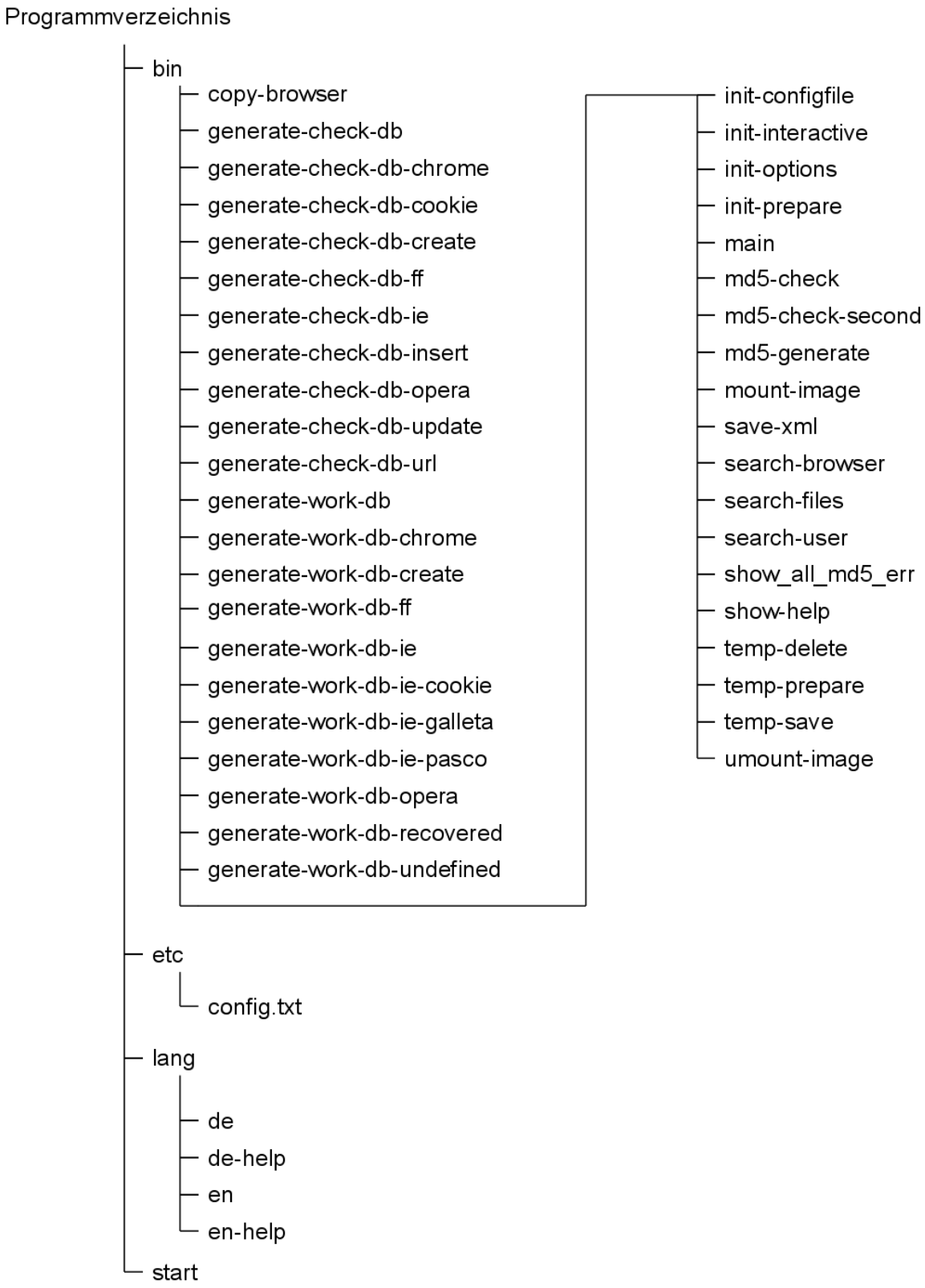


Abbildung 8: Struktur des Programmverzeichnises

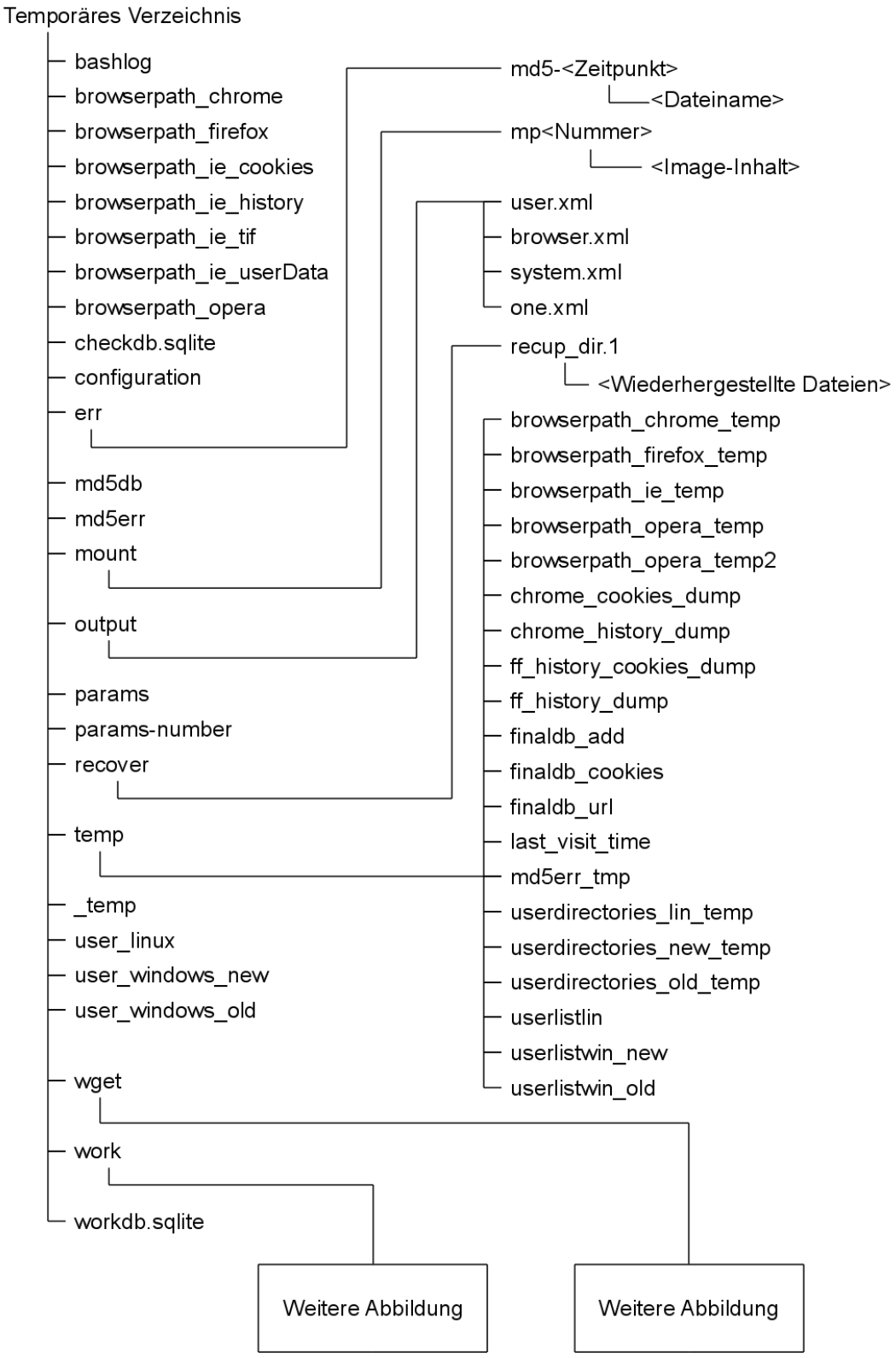


Abbildung 9: Temporäres Verzeichnis: Ordner- und Dateistruktur

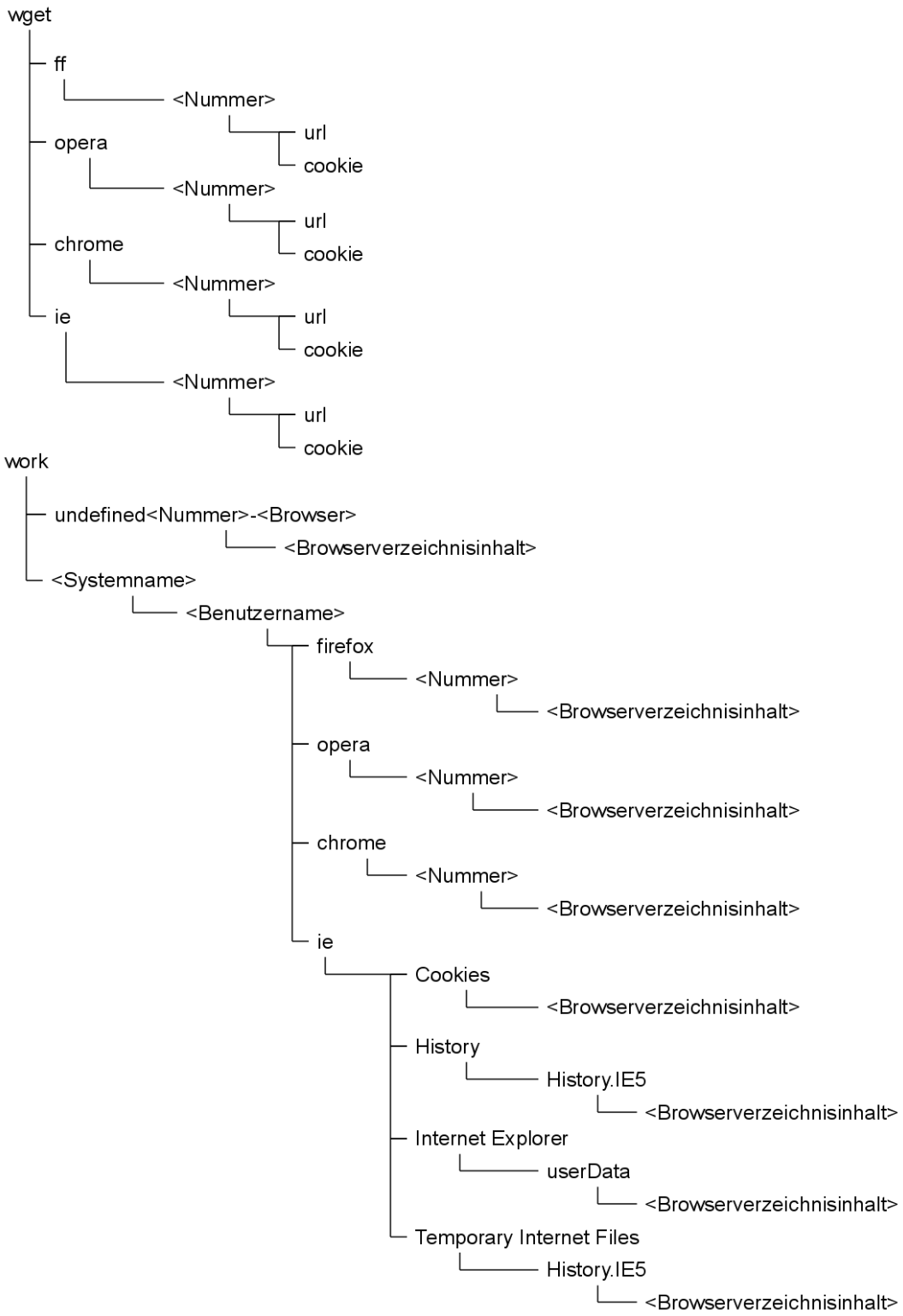


Abbildung 10: Temporäres Verzeichnis: Ordner “work” und “wget”

- Alle Dateien, welche mit “browserspath*” beginnen, enthalten Pfade zu den Browsers im eingebundenem Image, wobei jeder Typ als eigene Datei angelegt wurde.
- Die beiden Dateien, welche auf “.sqlite” enden, beinhalten die beiden erstellten Datenbanken. Die gefundenen Daten werden in der “workdb.sqlite” gespeichert. Die verarbeiteten und getesteten Einträge sind in der “checkdb.sqlite” zu finden. Durch diese strikte Trennung kann die zweite Datenbank auch als Ausgabe verwendet werden.
- In der Datei “configuration” ist die für den Ablauf notwendige Konfiguration gespeichert.
- Die Dateien “md5db” und “md5err” beinhalten MD5-Summen und Dateien, die als fehlerhaft erkannt wurden.
- In den Dateien “params” und “params-number” werden die am Programmstart übergebenen Werte und deren Anzahl gespeichert.
- Alle Dateien, welche mit “user*” beginnen, beinhalten die gefundenen Benutzernamen für das jeweils im zweiten Teil des Namens angegebene System.
- Die Datei “_tmp” dient als temporärer Speicherplatz für die Rückgabevariablen von “dialog”. Strukturbedingt konnte diese Datei nicht in den Unterordner “temp” gelegt werden.

Die Ordner im temporären Verzeichnis beinhalten folgende Daten:

- **err:**
  Dieser beinhaltet alle Dateien, bei denen während der Bearbeitung MD5-Fehler aufgetreten sind. Dazu wird zuerst ein Unterordner angelegt, in welchem der Zeitpunkt der Erkennung eingetragen wird.
- **mount:**
  In diesem Ordner wird für jede einzuhängende Partition ein Ordner mit einer laufenden Nummer angelegt. Anschließend wird diese dort eingebunden.
- **output:**
  In diesem Ordner werden die generierten XML-Dateien gespeichert, bevor sie in

den Ordner des Programmaufrufs kopiert werden. Im Backup des Arbeitsordners sind diese also ebenfalls vorhanden.

- **recover:**
  In diesem werden alle durch "photorec" wiederhergestellten Dateien gespeichert. Der Unterordner "recup_dir.1" wird dabei vom externen Programm erstellt.

- **temp:**
  Dieser beinhaltet alle für die Bearbeitung notwendigen temporären Dateien. Welches Skript welche Datei nutzt, wird im jeweiligen Unterpunkt von Kapitel 6.7 auf Seite 60 erwähnt.

- **wget:**
  Dieser Ordner beinhaltet einen Unterordner für jeden unterstützten Browser. Jede gefundene URL wird mittels "wget" abgerufen und die jeweilige Datei in einen Ordner mit laufender Nummer, welche auch in der "checkdb.sqlite" zu finden ist, gespeichert.

- **work:**
  In dieses Verzeichnis werden die der Datenquelle entnommenen Dateien für den jeweiligen Browser kopiert. Nicht zuordenbare Browser-Verzeichnisse werden in einen eigenen Ordner kopiert, dessen Dateiname sich aus "undefined" mit angehängter laufender Nummer und Browsertyp zusammensetzt (z.B.: undefind0-ie).

Um zu genaueren Informationen über den jeweiligen Datei-Inhalt zu gelangen, gibt es folgende Möglichkeit:

Auf der mitgelieferten CD befinden sich mehrere Sicherungen der Testdurchläufe. Diese können einfach mittels ZIP entpackt und die darin enthaltenen Dateien betrachtet werden.

6.4 Sprachen

Das erstellte Programm liefert die Möglichkeit, auf einfache Weise die Sprache zu wechseln. Dazu muss nur im Programmverzeichnis im Unterordner "etc" die Datei "config.txt" bearbeitet werden. In der Zeile "LANGFILE=" wird einfach die gewünschte Sprache als Kürzel eingetragen. Die dieser Arbeit beigelegte Version unterstützt Deutsch ("de") und

Englisch (“en”), sowohl für das eigentliche Programm als auch für die Hilfe.

Soll eine weitere Sprache hinzugefügt werden, besteht die beste Lösung darin, einfach eine bestehende Datei zu öffnen und unter einem neuen Sprachkürzel zu speichern. Anschließend kann diese einfach übersetzt werden. Dasselbe gilt auch für die Hilfedateien, die mit demselben Kürzel wie die Sprachdatei beginnen müssen und “-help” angehängt wird. Fehlt eine Sprachdatei, wird die Programmausführung nicht gestartet. Würde die Datei während der Ausführung gelöscht, so wird der eigentliche Programmablauf trotz auftretender Fehler in der Ausgabe nicht beeinflusst.

Das Ausgeben eines Textes unter Berücksichtigung der Sprache geschieht folgendermaßen:

```
1 echo "### ‘cat $progpath/lang/$langfile |grep "save-xml_START=" | ↩
    cut -d "=" -f2‘"
```

Zunächst liest dabei der Befehl “cat” die im Programmverzeichnis vorhandene Sprachdatei aus und reicht deren Inhalt an den Befehl “grep” weiter. Dieser sucht die Zeile mit dem jeweiligen sprachunabhängigen Signalwort (hier: “save-xml”) und übergibt diese an den Befehl “cut”, welcher den vorderen Teil aus dem String entfernt. Der auf diese Weise ausgelesene Sprachteil wird durch drei vorangestellte Rautezeichen erweitert und abschließend mittels “echo” auf der Konsole ausgegeben.
Die für das aufgeführte Beispiel zugehörigen Zeilen, für die deutsche und englische Sprache, wären folgende:

Deutsch (Datei: “<Programmverzeichnis>/lang/de”):

```
save-xml_START=Exportiere Datenbank in XML Struktur ...
```

Englisch (Datei: “<Programmverzeichnis>/lang/en”):

```
save-xml_START=Exporting database in XML structure ...
```

Wie aus dieser Auflistung auch gut ersichtlich ist, kann jede Zeile auf einfache Weise in eine neue Sprache übersetzt werden. Wenn in einer neuen Sprachdatei ein Eintrag

wie der angeführte fehlt, wird dies als leerer String vom Unterbefehl zurückgegeben und "echo" gibt dadurch nur die drei Rautezeichen aus. Die eigentliche Programmfunktion wird dadurch auch nicht beeinflusst oder gar unterbrochen.

6.5 Voraussetzungen

Damit die erstellte Anwendung erfolgreich funktioniert, müssen mehrere Voraussetzungen erfüllt sein, welche in diesem Kapitel besprochen werden.

Zunächst muss der User, welcher das Programm ausführt, die notwendigen Rechte zum Einbinden und Lösen von Images oder Festplatten besitzen. Dies könnte natürlich als User "root", der ja alle Rechte besitzt, erfolgen. Davon wird jedoch aufgrund der damit verbundenen Risiken dringend abgeraten, denn eventuelle Skriptfehler oder bösartige Dateien, die im Laufe der Bearbeitung heruntergeladen oder aus dem Image kopiert werden, könnten das gesamte System beschädigen. Das Programm soll daher nur mit normalen Benutzer-Rechten ausgeführt werden.

Eine mögliche Lösung dafür wäre, die Berechtigung für die Befehle "mount", "umount" und "losetup" mittels "sudo" an den aktuellen User zu übertragen. Dabei kann ebenso die Möglichkeit gesetzt werden, diese Berechtigungen ohne Root-Passwort durchzuführen. Um dies zu bewerkstelligen, muss auf einer aktuellen Linux-Distribution die Datei "/etc/sudoers" mit Hilfe des Programms "visudo" bearbeitet werden. In diese Datei wird anschließend folgende Zeile unter Punkt "# User privilege specification" eingefügt:

```
%sudo ALL=(ALL)NOPASSWD:/bin/mount,/bin/umount,/sbin/losetup
```

Damit berechtigt man alle Benutzer der Gruppe "sudo" dazu, die drei angeführten Befehle ohne Passwort auszuführen. Statt "%sudo" könnte auch die Gruppe des aktuellen Users oder der Loginname ohne "%" stehen. Ist dieser Eintrag erstellt, so muss man dem gewünschten Benutzer noch die Rechte für die Gruppe "sudo" geben. Dies kann durch den nachfolgenden Befehl für den User "msc-debian" erfolgen:

```
usermod -a -G sudo msc-debian
```

Anschließend muss der Benutzer neu eingeloggt werden.

Damit das Programm auch funktioniert, müssen neben der Berechtigung auch auch die benötigten externen Programme installiert sein. Wichtig ist zu erwähnen, dass sämtliche Programme unter einer freien Lizenz verfügbar sind. Außerdem handelt es sich bei den meisten dieser Programme um Standard-GNU-Tools, welche auf einem modernen Desktop-Linux automatisch installiert werden. Während der Programmierung wurde versucht, die Liste der Abhängigkeiten zu minimieren. Folgende fünfundzwanzig Abhängigkeiten werden allerdings benötigt:

- **awk:*** [35]
  Bei diesem Programm handelt es sich um ein Standard-GNU-Tool. Mit diesem lassen sich sehr umfangreiche Textoperationen ausführen. Diese Anwendung wird auch als eigene Skriptsprache bezeichnet.

- **cat:*** [36]
  Mit Hilfe dieses Befehls, welcher in den GNU-Coreutils enthalten ist, lässt sich unter anderem der Inhalt einer Textdatei gezielt auslesen.

- **cut:*** [37]
  Mit Hilfe dieses Befehls, welcher in den GNU-Coreutils enthalten ist, lässt sich ein übergebener String einfach manipulieren. Für einfache Operationen ist dieser Befehl problemloser und schneller als “awk”.

- **cp:*** [38]
  Mit Hilfe dieses ebenfalls in den GNU-Coreutils enthaltenem Befehl lassen sich Dateien im Dateisystem kopieren.

- **dialog:** [39]
  Dieses Programm ist für eine einfache grafische Ausgabe in der Kommando-Zeile zuständig. Damit lassen sich auch interaktive Abfragen und Ausgaben erstellen.

- **fdisk:** [40]
  Bei dieser Anwendung handelt es sich um ein Tool zur Festplattenverwaltung. Es lassen sich damit auch Partitionen und deren Parameter aus einem Image auslesen.

- **find:*** [41]
  Mit Hilfe dieses Programms lässt sich ein Dateisystem nach Dateien durchsuchen. Dazu können auch reguläre Ausdrücke oder Namensteilbereiche verwendet werden.

- **galleta:** [7]
  Das Tool ist Teil des ODESSA-Framework, in welchem auch "pasco" enthalten ist. Es lassen sich mit dessen Hilfe Cookies aus der InternetExplorer-History auslesen.

- **grep:** [42]
  Dieses Programm durchsucht übergebene Dateien oder Texte nach definierbarem Inhalt und gibt gefundene Zeilen aus. Zum Suchen können auch reguläre Ausdrücke verwendet werden.

- **tee:*** [43]
  Dieser Befehl leitet die gesamte Ausgabe an die Befehlszeile und an eine definierte Datei weiter. Dadurch kann zum Beispiel eine Log-Datei erstellt werden.

- **losetup:** [44]
  Bei dieser Anwendung handelt es sich um ein Tool des "util-linux-ng"- Framework. Darüber lässt sich ein Image über ein Loop-Device einhängen und schließen sowie eine normale Festplatte ansprechen.

- **ls:*** [45]
  Dieses Programm zeigt den Inhalt eines Ordners auf der Standard-Ausgabe an. Es lassen sich damit auch Dateiattribute auslesen.

- **md5deep:** [46]
  Dieses Tool erstellt MD5-Summen von mehreren Dateien, hinterlegt diese in eine Datei und kann anschließend neue Dateien gegen die existierenden vergleichen. Auch ein rekursives Bearbeiten von Ordnern ist möglich.

- **md5sum:*** [47]
  Ähnlich wie md5deep erstellt diese Anwendung MD5-Summen von einzelnen Dateien. Ein rekursives Bearbeiten von Ordnern ist dabei allerdings nicht direkt möglich.

- **mkdir:*** [48]
  Durch diesen Befehl lässt sich ein neuer Ordner im Dateisystem erstellen.

- **mount:** [44]
  Das in den "util-linux-ng" enthaltene Tool "mount" ist für das Einbinden von Partitionen in das Dateisystem zuständig. Mit Hilfe von "losetup" kann auch ein RAW-Image eingebunden werden.

- **mv:*** [49]
  Durch dieses Programm können Dateien auf der Festplatte verschoben oder umbenannt werden.

- **pasco:** [7]
  Dieses Tool ist ebenfalls Teil des ODESSA-Framework. Mit dessen Hilfe lässt sich der Inhalt der "index.dat" vom InternetExplorer in einer lesbaren Form aufbereiten.

- **photorec:** [13]
  Mit dieser Anwendung können gelöschte Dateien von einer Festplatte wiederhergestellt werden. Dies funktioniert auch, wenn das Dateisystem unwiederbringlich beschädigt ist.

- **rm:*** [50]
  Dieses Tool dient zum Löschen von Dateien und Ordnern.

- **rsync:** [51]
  Dieses Programm dient zum inkrementellen Kopieren von Dateien. Das Ziel oder die Quelle kann dabei auch auf einem anderen Rechner liegen. Kopiert wird nur der geänderte Inhalt der Dateien, wobei dies durch Checksummen sichergestellt wird.

- **sed:*** [52]
  Diese Anwendung dient zum automatisierten und nicht interaktiven Bearbeiten von Textdateien oder Strings. Inhalte können damit zum Beispiel einfach ausgetauscht werden.

- **sqlite3:** [53]
  Durch dieses Programm können SQLite-Datenbanken mittels SQL-Befehlen erstellt, gelesen und geändert werden.

- **touch:*** [54]
  Durch dieses Tool kann eine leere Datei angelegt oder der Zeitstempel einer bestehenden geändert werden.

- **umount:** [44]
  Mit Hilfe dieser Anwendung, die Bestandteil der “linux-util-ng”- Bibliothek ist, kann die Einbindung von Festplatten gelöst werden.

- **zip:** [55]
  Dieses Programm dient zum Packen von Dateien und Ordnern in ein standardisiertes “ZIP”-Archiv. Ein Entpackvorgang ist ebenfalls möglich.

Bei allen Programmen, welche mit “*” markiert wurden, handelt es sich um GNU-Tools. Diese sind bis auf “awk”, “sed“ und “find” in den “coreutils” [56] enthalten und müssen daher nicht extra installiert werden.

Tests auf mehreren GNU|Linux-Systemen haben gezeigt, dass nach einer Installation nur folgende der aufgelisteten Programme zusätzlich installiert werden müssen: “galleta”, “md5deep”, “pasco”, “photorec”, “rsync”, “sqlite3” und “zip”.

6.6 Programmablauf

Nachdem in den vorherigen Kapiteln grundlegende Hintergrundinformationen beschrieben wurden, widmet sich dieses ganz dem eigentlichen Programmablauf. In der Abbildung 11 auf der nächsten Seite ist der Ablauf des Programms nummeriert dargestellt. Die dort angeführten Ordnernamen, die mit “<name>” beginnen, wurden aufgrund von Platzersparnis abgekürzt. Der volle Name setzt sich aus dem des aufrufenden Skripts und dem Kürzel zusammen. So bedeutet zum Beispiel “<name>-chrome”, dass der eigentliche Dateiname “generate-work-db-chrome” lautet.

Zur Ausführung wird das Programm über das Skript “start” aufgerufen. Dieses sucht zuerst die Programmumgebung, testet, ob alle benötigten Abhängigkeiten installiert sind, und speichert zusätzlich die übergebenen Startparameter für eine spätere Ausführung in eine Datei. Anschließend wird über das Skript “main” aufgerufen.

Das Skript “main” entspricht dem Skript, welches nur die Funktion besitzt, alle Unterskripte in der richtigen Reihenfolge aufzurufen und bei Fehlern einen kompletten Absturz zu verhindern. Dieser Ablauf ist folgender:

1. Analyse der übergebenen Parameter durch das Skript “init-prepare”. Davon abhängig wird einer der für die Initialisierung zuständigen Skripte gestartet. Die

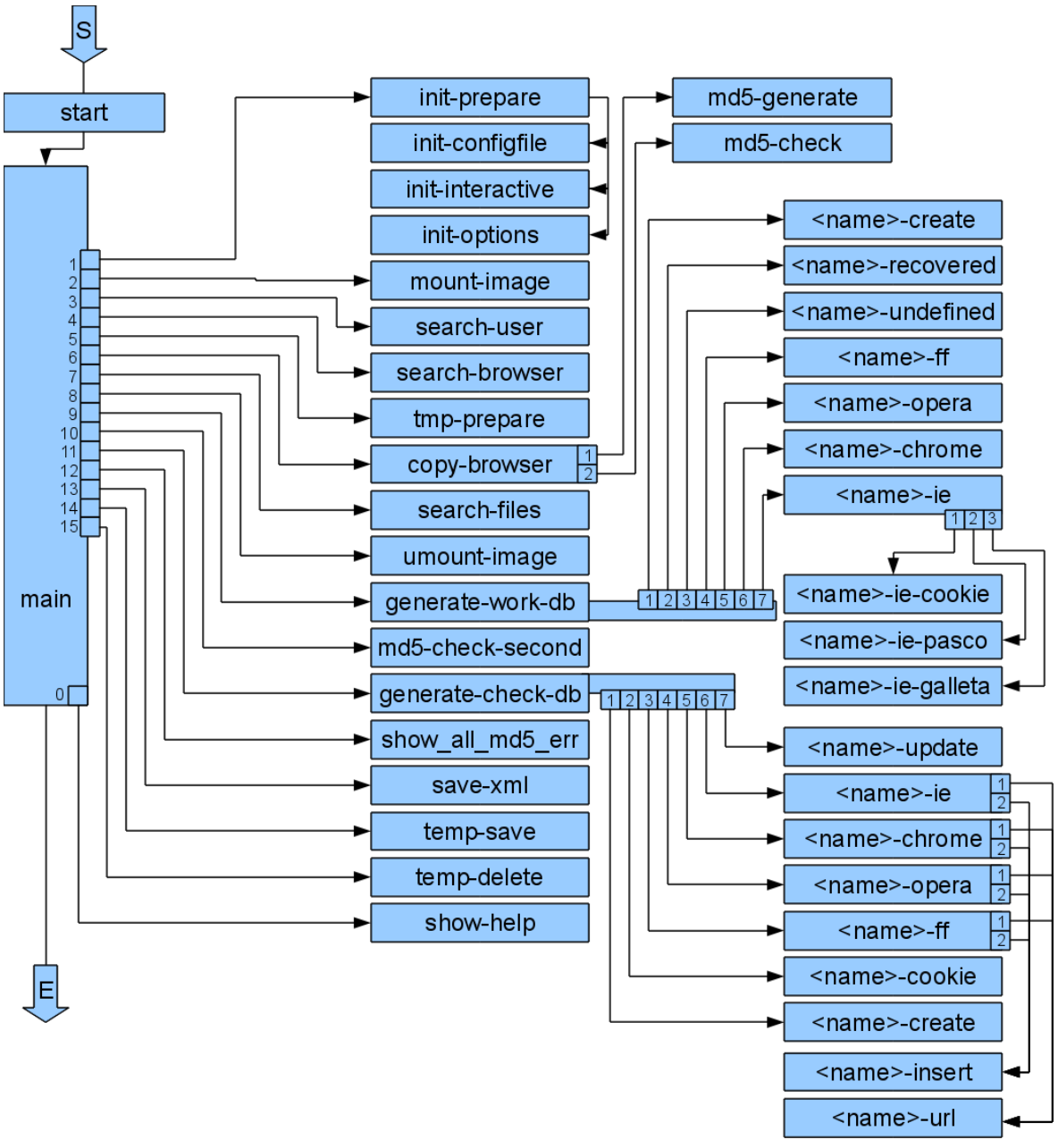


Abbildung 11: Programmablauf

möglichen Skripte sind dabei “init-configfile”, “init-options” und “init-interactive”. Jedes von ihnen wertet die jeweils übergebenen Parameter aus und speichert die Ergebnisse in die temporäre Konfigurationsdatei. Beim Programmaufruf besteht die Möglichkeit zwischen dem interaktiven Modus ohne Angabe von Daten, dem Start mittels übergebener Konfigurationsdatei oder durch direktes Angeben der einzelnen Parameter zu wählen. Die genau beschriebenen Möglichkeiten zum Programmaufruf sind in Kapitel 7.2 auf Seite 107 zu finden.

2. Das übergebene Image oder die Quelldatei werden nun über “mount-image” nur lesend eingebunden.

3. Anschließend werden über das Skript “search-user” alle im Image enthaltenen Benutzer inklusive zugehörigem Betriebssystem gesucht. Unterschieden wird dabei zwischen Linux, WindowsOLD (XP und 2000) und WindowsNEW (Vista, 7).

4. Im Anschluss daran wird das Image über das Skript “search-browser” nach allen enthaltenen Browser-Verzeichnissen durchsucht und anhand des Pfades den zuvor gefundenen Usern zugeordnet. Bei nicht erkanntem User wird dieser als “undefined” mit laufender Nummer bezeichnet.

5. Darauffolgend wird nun mit Hilfe des Skriptes “tmp-prepare“ das temporäre Verzeichnis für den Kopiervorgang vorbereitet. Dazu werden etwa im Unterordner “work” für jedes System, für jeden User und für jeden Browser Ordner angelegt.

6. Ist dies erledigt, beginnt das Skript “copy-browser” den Kopiervorgang der gefundenen Ordner in die zuvor angelegten Ordner mittels des Programms “rsync” zu kopieren. Zusätzlich werden pro Ordner die Skripte “md5-generate” und “md5-check” aufgerufen. Ersteres erstellt für jede neue Datei einen Eintrag in die MD5-Datenbank. Das andere testet die kopierten Dateien auf deren MD5-Summe. Gibt es dabei Fehler, wird die betroffene Datei in den Ordner “err” verschoben und eine Warnung ausgegeben.

7. Nach dem Abschluss des Kopiervorganges wird das Image über das Skript “search-files” nach gelöschten Dateien durchsucht. Dies geschieht über das Programm “photorec” und kann je nach Größe des Images oder der Festplatte einige Zeit beanspruchen.

8. Anschließend wird die Einbindung des Images über das Skript “umount-image” gelöst. Sollte es sich dabei um einen über USB angeschlossenen Speicher oder einen anderen Wechseldatenträger handeln, kann dieser jetzt vom Benutzer entfernt werden.

9. Mit diesem Schritt beginnt nun das Auslesen der einzelnen kopierten Browserordner. Das Skript “generate-work-db” steuert dabei ähnlich wie “main” eigene Unterskripte für den jeweiligen Browser an. “generate-work-db-create” erstellt dabei die benötigte Datenbank. Alle anderen tragen die von ihnen gefundenen Daten in diese Datenbank ein. Für den InternetExplorer mussten weitere Unterskripte angelegt werden, damit bei Abstürzen der externen Programme “pasco” und “galleta” nur die jeweilige Datei nicht ausgelesen wird, das restliche Programm aber weiterhin funktioniert.

10. Wurde die erste Datenbank erstellt, prüft das Skript “md5-check-second”, ob bei der Bearbeitung auch keine Dateiänderungen vorgenommen wurden. Bei Problemen werden diese ebenfalls in den Ordner “err” verschoben und es wird ein entsprechender Hinweis ausgegeben.

11. Nachfolgend startet das Skript “generate-check-db” die Auswertung der zuvor erstellten Datenbank. Dazu werden ebenfalls wieder Unterskripte benötigt. So erstellt zuerst “generate-check-db-create” die neue Datenbank. Anschließend werden alle Cookies mit Hilfe von “generate-check-db-cookies” aus der alten in die neue übertragen. Ist dies erledigt, werden alle Browser über das jeweils zugehörige Skript bearbeitet. Mit “generate-check-db-url” wird die jeweils gefundene URL heruntergeladenen. Durch das Skript “generate-check-db-insert” wird der jeweilige aus der alten Datenbank entnommener Eintrag inklusive der Testergebnisse in die neue eingetragen. Wurden alle Daten übernommen und getestet, berechnet “generate-check-db-update” die einzelnen Browsersitzungen.

12. Im nächsten Schritt werden nun erneut gefundene MD5-Fehler über das Skript “show_all_md5_err” ausgegeben.

13. Darauffolgend erstellt das Skript “save-xml” die gewünschten XML-Dateien durch Auslesen der neueren Datenbank.

14. Ist dies erledigt, ist der eigentliche Programmablauf beendet und der Arbeitsordner kann über das Skript “temp-save” in eine ZIP-Datei gesichert werden.

15. Zum Programmabschluss wird der temporäre Ordner gelöscht.

Damit wäre ein kompletter Programmdurchlauf abgeschlossen und das Skript “main” beendet sich selbst. Die nicht erwähnte Datei “show-help” kommt nur zum Einsatz, wenn als Startparameter “-h” oder “–help” übergeben wurde. Wird diese aufgerufen, so werden alle möglichen Startoptionen und Parameter angegeben.

Informationen zu den Speicherorten der einzelnen in diesem Kapitel erwähnten Dateien können in Kapitel 6.3 auf Seite 39 nachgelesen werden. Die genaue Funktionsweise jedes erwähnten Skripts wird im nachfolgenden Kapitel 6.7 besprochen. Getestete Programmdurchläufe werden in den Unterpunkten von Kapitel 7 auf Seite 104 besprochen, Stärken (7.6) und Schwächen (7.7) in der aktuellen Implementierung erwähnt. Zusätzlich wird im Unterkapitel 7.5 auf Seite 116 auch auf die getesteten Festplatten-Images eingegangen.

6.7 Sub-Programme

In diesem Kapitel werden auf die einzelnen Skripte des Programms genauer besprochen. Dabei wird der jeweilige Zweck erläutert und auf die Funktionsweise eingegangen. Zusätzlich werden die jeweiligen Ein- und Ausgaben, sowie die benötigten temporären Dateien behandelt.

Die Sortierung der besprochenen Skripte basiert auf dem Ablauf des Programms, wie es in Kapitel 6.6 auf Seite 56 besprochen wurde. Auf der dieser Arbeit beiliegenden CD ist auch der Quelltext der einzelnen Skripte zu finden.

Die Beschreibung wird nun mit dem Start-Skript begonnen, das vom jeweiligen Anwender aufgerufen wird.

6.7.1 Skript: start

Dieses Skript wird vom Anwender direkt aufgerufen und befindet sich daher auch direkt im Hauptordner des Programms. Es ist für folgende Punkte zuständig:

- **Suchen des Programmpfades:**
  Sollte dieses Skript in einen anderen Ordner kopiert werden, besitzt es die Möglichkeit, den eigentlichen Programmordner selbstständig zu finden. Dadurch entfällt auch eine fixe Installation.

- **Überprüfen der abhängigen Programme:**
  Es wird getestet, ob alle in Kapitel 6.5 auf Seite 52 erwähnten Programme auf dem Betriebssystem installiert sind. Bei Fehlern wird das Programm mit der entsprechenden Meldung beendet.

- **Anlegen des temporären Ordners:**
  Es wird der Ordner für die aktuelle Bearbeitung angelegt. Dorthin wird auch die temporäre Konfigurationsdatei mit Standardwerten kopiert. Anschließend werden noch benötigte temporäre Dateien erstellt.

- **Speichern der übergebenen Parameter:**
  Die übergebenen Parameter werden in die definierte Parameterdatei der zuvor erstellten Ordner gespeichert. Dies gilt ebenso für deren Anzahl.

- **Definieren vom Display:**
  Sofern in der fixen Konfiguration "auto" für das Display angegeben wurde, wird getestet, ob "dialog" (nur Konsole) oder "Xdialog" (grafische Fenster) möglich ist. Dies wird auch für alle späteren Skripte definiert.

Wurden alle bearbeiteten Punkte abgeschlossen, so wird das Skript "main" für die Fortsetzung des Programms aufgerufen. Dabei wird zusätzlich auch ein Loggen der Ausgabe mittels "tee" aktiviert. Der dazu verwendete Befehl lautet:

```
bash $progpath/bin/main $tempdir/$configfile | tee -a ↩
   $tempdir/bashlog
```

Es wird also im Programmpfad das Skript "main" mit Übergabe der temporären Konfigurationsdatei aufgerufen. Zusätzlich wird die gesamte Ausgabe an "tee" weitergeleitet. Dieses Programm speichert alles in die Datei "bashlog" und gibt dieselben Zeilen auch auf der Konsole aus.

6.7.2 Skript: main

Dieses Skript steuert nur alle weiteren Unterskripte an und testet nach Beendigung des jeweiligen Bereiches, ob das Programm vorzeitig, etwa aufgrund eventueller Fehler, beendet werden soll. Die Reihenfolge der Aufrufe wird in Kapitel 6.6 auf Seite 56 besprochen.

Zusätzlich wird bereits mit der Begrüßung die Mehrsprache-Ausgabe, wie sie in Kapitel 6.4 auf Seite 50 erwähnt wurde, eingesetzt.

6.7.3 Skript: init-prepare

Das Skript "init-prepare" soll herauszufinden, welche Art von Parametern übergeben wurden. Dabei werden alle übergebenen Optionen gegen definierte Signalworte geprüft. Eines der folgenden Skripte kann dabei aufgerufen werden:

- **"init-interactive":**
  Dies wird genutzt, wenn kein Parameter übergeben wurden.
- **"init-configfile":**
  Dieses Skript wird gestartet, wenn als Parameter "-c" oder "–configfile=<name>" übergeben wurde.
- **"show-help":**
  Durch dessen Aufruf, welcher mit "-h" oder "–help" erfolgen kann, wird die Hilfsdatei in der eingestellten Sprache dargestellt.
- **"init-options":**
  Sollte keiner der vorher erwähnten Parameter gefunden werden, so wird dieses Skript für eine genauere Auswertung gestartet.

Sobald einer der erwähnten Parameter gefunden wurde, werden alle weiteren Punkte für eine optionsbasierte Konfiguration nicht mehr beachtet. Die genaue Auflistung aller Startparameter ist in Kapitel 7.2 auf Seite 107 zu finden.

Wichtig zu erwähnen ist auch, dass die bei den Parametern weiter hinten stehenden Optionen immer die größere Priorität haben. So wird etwa bei "-c <name> -h" nur die Hilfe aufgerufen, nicht aber die Konfiguration mittels übergebener Datei.

6.7.4 Skript: init-configfile

Über dieses Skript wird die beim Start übergebene Konfigurationsdatei ausgelesen, anschließend werden die Werte getestet und auf die temporäre Konfiguration angewendet. Das Vorgehen ist dabei folgendes:

1. Es werden erneut alle Startwerte nach “-c” oder “–configfile” durchsucht. Wurde einer der beiden gefunden, kann der String mit dem Pfad auf die zugehörige Variable angewendet werden.

2. Es wird getestet, ob die übergebene Datei existiert und lesbar ist.

3. Anschließend wird die Datei zeilenweise ausgelesen. Jede Zeile wird auf einen möglichen Startparameter durchsucht. Wurde einer gefunden, wird dieser in eine der vorgesehenen Variablen gespeichert. Dies geschieht so lange, bis alle Zeilen eingelesen wurden. Sollte ein Wert doppelt existieren, überschreibt der spätere den ersten.

4. Die gelesenen Daten werden anschließend auf deren Plausibilität hin getestet und direkt in die temporäre Konfigurationsdatei mittels “sed” eingetragen. Die bis zu diesem Zeitpunkt dort eingetragenen Standardwerte werden sozusagen ersetzt.

Wurden alle Optionen ausgelesen und in die temporäre Konfigurationsdatei eingetragen, ist die Aufgabe dieses Skripts abgeschlossen.

Eine Konfigurationsdatei kann beispielsweise folgenden Aufbau besitzen:

```
1  ### configfile
2  SOURCE=/home/user/MSC-Debian.img
3  USER=all
4  BROWSER=firefox,chrome
5  SEARCHFILES=no
6  CHECK=yes
7  COOKIES=yes
8  OUTPUT=all
9  SAVETEMP=yes
10 INTERACTIVE=no
```

Bei dieser Datei wird das Image “MSC-Debian.img” bearbeitet. Dabei werden alle Benutzer gesucht, die Browser Firefox und Chrome abgearbeitet, keine gelöschten Dateien gesucht, die gefundenen Daten inklusive Cookies getestet, alle möglichen XML-Ausgabedateien erstellt und temporäre Ordner gesichert. Die genauen Möglichkeiten und benötigen Werte werden in Kapitel 7.2 auf Seite 107 beschrieben.

6.7.5 Skript: init-options

Werden bei einem Programmaufruf Parameter direkt übergeben, welche nicht “-c”, “-h”, “–configfile=*” oder “–help” enthalten, wird dieses Skript ausgeführt. Der Ablauf ist dabei folgender:

- Es werden alle übergebenen Parameter der Reihe nach mit alle gültigen Werten vergleichen.
- Wurde ein Parameter erfolgreich erkannt, wird die ausgelesene Option in eine lokale Variable gespeichert.
- Bei mehrfach erkannten identischen Parametern überschreibt, gleich wie im Skript “init-configfile“, der zuletzt erkannte Werte die zuvor definierten. Dies geschieht durch das Ersetzen des Inhalts der Variable.
- Sind alle Felder abgearbeitet, werden alle Variablen verschiedenen Tests auf Plausibilität hin unterzogen. Fehlerhafte Optionen werden mit den Standardwerten für einen möglichst großen Funktionsumfang ersetzt.
- Nach den Tests werden alle Optionen direkt in die temporäre Konfigurationsdatei mittels des Befehls “sed” eingetragen.

Ein gültiger Programmaufruf ist folgender:

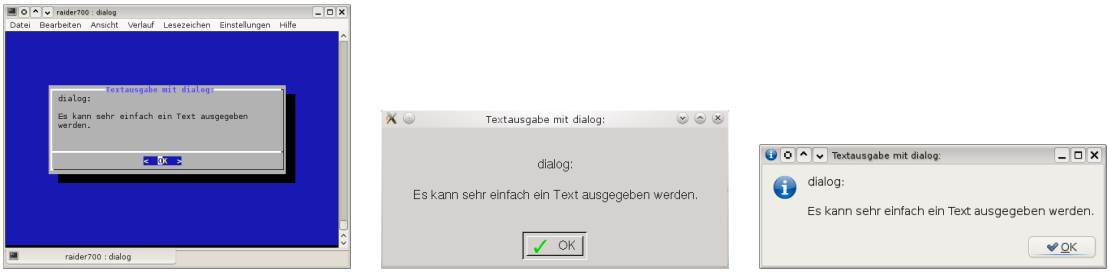
```
# lange Schreibweise:
bash start --source=/home/user/MSC-Debian.img --user=all ↩
   --browser=chrome,opera --searchfiles=yes --check=yes ↩
   --cookies=yes --output=one --savetemp=yes --interactive=no
# kurze Schreibweise:
bash start -s /home/user/MSC-Debian.img -u all -b chrome,opera ↩
   -sf yes -ch yes -co yes -o one -st yes -i no
```

Hier wird bei beiden Befehlen das Image “MSC-Debian.img” für die Auswertung genutzt. Dabei werden alle gefundenen Benutzer verwendet, die Browser Chrome und Opera abgearbeitet, nach gelöschten Dateien gesucht, die gefundenen Daten inklusive Cookies getestet, das Gefundene in einer gleichwertigen XML-Datei exportiert und das temporäre Verzeichnis gesichert. Die genauen Möglichkeiten und jeweils benötigen Werte sind in Kapitel 7.2 auf Seite 107 zu finden.

6.7.6 Skript: init-interactive

Werden bei einem Programmstart keine Optionen übergeben, wird dieses Skript für die Abfrage der einzelnen Parameter von “init-prepare” gestartet. Dieses Skript bietet dabei die Möglichkeit, alle Optionen interaktiv und auch grafisch abzufragen.

Für die grafische Ausgabe werden folgende Programme unterstützt:



(a) dialog (b) xdialog (c) gdialog

Abbildung 12: Ausgaben mit dialog, Xdialog und gdialog

- **dialog:** [39]
  Visualisiert die Ausgabe grafisch auf der Konsole mit Hilfe der “ncurses” Bibliothek [57]. Der Aufruf und der gewünschte Text erfolgen dabei über die Konsole. Eine Informationsbox kann zum Beispiel einfach mit folgendem Befehl aufgerufen werden:

```
dialog --title "Textausgabe mit dialog:" --msgbox ↩
   "dialog:\n\nEs kann sehr einfach ein Text ausgegeben ↩
   werden." 10 50
```

Die daraus resultierende Ausgabe ist in Abbildung 12(a) auf der vorherigen Seite zu sehen.

- **Xdialog:** [58]
  Stellt Ausgabebefehle über das GTK-Framework [59] grafisch dar. Die Befehlsmöglichkeiten sind dabei zu denen von "dialog" identisch. Die Befehlszeile von dem im vorherigen Punkt aufgeführten Beispiel kann, bis auf das Ersetzen von "dialog" durch "Xdialog", direkt übernommen werden. In Abbildung 12(b) auf der vorherigen Seite ist die Ausgabe des Beispiels zu sehen.

- **gdialog:** [60]
  Als eine Alternative zu "Xdialog" kann auch "gdialog" genutzt werden. Dies ist notwendig, da manche Distributionen das andere Programm nicht mehr anbieten. Die Befehle sind ebenfalls zu "dialog" kompatibel. In neueren Distributionen wird die Funktionalität durch "zenity", der neuen Standardausgabe für Gnome, realisiert. Dabei gibt es jedoch noch vereinzelt Darstellungsfehler. Deshalb musste zum Beispiel der Auswahldialog für Dateien mit einer Alternative implementiert werden. Kann das jeweilige System den erwähnten Dialog nicht öffnen, erscheint eine normale Eingabezeile. Die Ausgabe der in den vorherigen Punkten erwähnten Beispiele ist in Abbildung 12(c) auf der vorherigen Seite zu sehen.

Das Skript wurde so implementiert, dass die gewünschte Ausgabe in der Konfigurationsdatei eingestellt werden kann. Dort kann auch eine automatische Erkennung mittels der Option "auto" aktiviert werden. Der Ort und der Inhalt der Datei sind in Kapitel 6.3.1 auf Seite 43 zu finden.

Wichtig zu erwähnen ist, dass nur das Programm "dialog" zwingend benötigt wird. Die beiden anderen Programme werden nur dann aktiviert, wenn sie installiert sind und eine grafische Ausgabe möglich ist. Wurde zum Beispiel in der Konfiguration "Xdialog" eingetragen aber es steht grafische Ausgabe zur Verfügung, so wird automatisch "dialog" genutzt.

Abbildung 13 auf Seite 107 zeigt den Auswahldialog für den interaktiven Start. Die Funktionsweise der einzelnen angeführten Punkte wird in Kapitel 7.2 auf Seite 107 beschrieben.

6.7.7 Skript: mount-image

Mit Hilfe dieses Skripts können die übergebenen Datenquellen analysiert und in das Dateisystem nur lesend eingehängt werden. Zunächst wird die angegebene Datei mit Hilfe des Programms “fdisk” und den Startoptionen “-ul” analysiert. Dabei gibt es folgende drei Möglichkeiten:

- **Partition:**
  Handelt es sich um eine reine Partition, so wird dies in eine Variable eingetragen und der Vorgang zum Einbinden direkt gestartet.

- **Device:**
  Sollte es sich bei der Datenquelle um eine Festplatte oder ein Festplattenimage handeln, müssen zunächst die Positionen der einzelnen enthaltenen Partitionen erfasst werden. Anschließend können diese der Reihe nach eingebunden werden.

- **Nichts:**
  Handelt es sich bei der angegebenen Datei um keine der beiden erwähnten Arten, wird die Programmausführung mit einer Fehlermeldung abgebrochen.

Sollte es sich bei der Datei um ein komprimiertes Image handeln, muss dieses vor der Verwendung in ein RAW-Format umgewandelt werden. Für diesen Zweck bietet die unter einer freien Lizenz stehende Virtualisierungssoftware “qemu” [61] eigene Programme wie “qemu-img” an.

Damit eine komplette Festplatte gemountet werden kann, müssen zunächst die Positionen der einzelnen Partitionen ermittelt werden. Dies erfolgt über den so genannten Offset, welcher aus der Ausgabe von “fdisk” berechnet werden kann. Die Ausgabe für den Befehl “/sbin/fdisk -ul MSC-Debian.img” ist folgende:

```
You must set cylinders.
You can do this from the extra functions menu.

Disk MSC-Debian.img: 0 MB, 0 bytes
255 heads, 63 sectors/track, 0 cylinders, total 0 sectors
Units = sectors of 1 * 512 = 512 bytes
```

```
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00074687

         Device Boot      Start         End      Blocks   Id  System
MSC-Debian.img1   *        2048    11223039     5610496   83  Linux
Partition 1 does not end on cylinder boundary.
MSC-Debian.img2        11225086    41940991    15357953    5  ↩
    Extended
Partition 2 has different physical/logical endings:
     phys=(1023, 254, 63) logical=(2610, 180, 2)
MSC-Debian.img5        11225088    12638207      706560   82  ↩
    Linux swap / Solaris
MSC-Debian.img6        12640256    41940991    14650368   83  Linux
```

Daraus lassen sich folgende Informationen ablesen:

- Es gibt in diesem Image vier Partitionen, wobei eine davon eine erweiterte Partition ist und eine andere für die Auslagerung des Hauptspeichers zuständig war (SWAP). Der Inhalt der SWAP-Partition wird vom erstellten Programm nur ausgewertet, wenn die Suche nach gelöschten Dateien aktiviert ist.

- Die einzelnen Partitionen starten bei den Sektoren 2048, 11225086, 11225088 und 12640256.

- Die Sektorgröße für dieses Image beträgt 512 byte.

- Es handelt sich bei den nutzbaren Partitionen um ein Linux-Dateisystem (ext3, reiserfs, ...). Alternativ könnte dort auch “W95 FAT32” oder “HPFS/NTFS” stehen.

Der Offset berechnet sich aus dem Multiplikat des Startsektor mit der Sektorgröße. Für dieses Beispiel wären das für die nutzbaren Partitionen folgende Werte :

1. MSC-DEBIAN.img1: 2048 * 512 = 1048576

2. MSC-DEBIAN.img6: 12640256 * 512 = 6471811072

Mit Hilfe dieser Werte kann nun der eigentliche Einbindungsvorgang durchgeführt werden. Dieser ist identisch mit dem eines Images einer Partition. Zunächst wird ein Ordner inklusive laufender Nummer im temporären Verzeichnis erstellt und der Dateisystemtyp festgestellt. Anschließend wird die Partition über den nachfolgenden Befehl nur lesend eingebunden:

```
sudo mount <Quell-Datei> <Zielverzeichnis> -t <Dateisystem> -o ↩
    ro,loop,offset\=<Berechneter Wert>
```

Im Programm wird für "<Dateisystem>" entweder "vfat", "ntfs-3g" oder "auto" angegeben. Die automatische Erkennung von Linux-Dateisystemen funktioniert für alle für das System möglichen Dateisysteme problemlos. Die Erkennung würde auch für alle aktuellen NTFS- und FAT-Arten funktionieren, doch dafür stehen meist auch mehrere Möglichkeiten zur Verfügung, weshalb diese zur eventuellen Problemvermeidung direkt angegeben werden.

Wichtig ist auch die Option "ro", da durch diese die Einbindung nur lesend erfolgt. Da dieser Schreibschutz auf Kernel-Ebene erfolgt, kann ein Schreibzugriff nur durch ein erneutes Einbinden erfolgen und ist daher für die restliche Programmausführung sichergestellt.

Der Punkt "loop" und "offset" aktiviert ein Einhängen der Quelldatei mit dem im Offset angegebenen bytebasierenden Versatz in ein Loop-Device. Dies erfolgt mittels "losetup" vor dem eigentlichen Mountvorgang.

6.7.8 Skript: search-user

Dieses Skript durchsucht alle eingebundenen Partitionen nach vorhandenen Benutzern. Dies geschieht unter Berücksichtigung des vorhandenen Betriebssystems. Das Vorgehen dabei ist folgendes:

- Suchen nach Dateien mit dem Namen "passwd". In einer solchen Datei werden in einem Linux-System alle Benutzer mit deren Standardordnern und weiteren Informationen gespeichert. Werden mehrere Dateien gefunden, so werden alle darin enthaltenen Benutzer in einer Liste kombiniert.

- Suchen nach einem Ordner mit dem Namen “Users”. Darin werden in Windows Vista und Windows 7 Unterordner mit den Namen der einzelnen Benutzer erstellt. Diese Namen werden ausgelesen und in eine Liste eingetragen. Gibt es mehrere Ordner mit diesem Namen, so werden die Benutzer kombiniert.
- Suchen nach einem Ordner mit dem Namen “Dokumente und Einstellungen” oder “Documents and Settings”. In diesen Ordnern werden unter Windows 2000 und Windows XP die Ordner für die jeweiligen Benutzer angelegt. Diese werden gleich wie bei “Users” in eine eigene Liste eingetragen.
- Alle erstellten Listen werden in eine Datei im temporären Ordner kopiert. Zusätzlich wird dabei für jeden darin enthaltenen Benutzer ein Ordner im zum jeweiligen Systemtyp gehörenden Ordner angelegt. Genaueres zum Aufbau ist in Kapitel 6.3.2 auf Seite 45 zu finden.

Die jeweilige Suche erfolgt über den Befehl “find”. Die nachfolgenden vier Befehle sind die jeweils genutzten Suchbefehle:

```
1 find $tempdir/mount -name passwd 2> /dev/null > ↩
    $tempdir/temp/userdirectories_lin_temp
2 find $tempdir/mount -type d -name Users 2> /dev/null | grep ↩
    mp[0-9]/Users > $tempdir/temp/userdirectories_new_temp
3 find $tempdir/mount -type d -name "Dokumente und Einstellungen" ↩
    2> /dev/null | grep mp[0-9]/Dokumente\ und\ Einstellungen > ↩
    $tempdir/temp/userdirectories_old_temp
4 find $tempdir/mount -type d -name "Documents and Settings" 2> ↩
    /dev/null | grep mp[0-9]/Documents\ and\ Settings >> ↩
    $tempdir/temp/userdirectories_old_temp
```

Bei den Befehlen mit enthaltenem “grep” wird gleichzeitig eine Kontrolle für die Position der Ordner durchgeführt. Die Ordner, die auf der Festplatte direkt im Wurzelverzeichnis liegen, sind gültig, alle anderen können herausgefiltert werden.

Sollen für Windows 2000 und Windows XP mehr als nur die beiden Sprachen Deutsch und Englisch unterstützt werden, müssen weitere “find”-Befehle hinzugefügt werden. Dazu kann die letzte der angegebenen Zeilen kopiert und die Ordnernamen durch die für

die jeweilige Sprache genutzten Namen ersetzt werden. Wichtig ist das Zeichen “»”, um die Liste nicht zu ersetzen, sondern den Inhalt an die bestehenden Einträge anzuhängen.

6.7.9 Skript: search-browser

Dieses Skript funktioniert ähnlich wie das zuvor erwähnte “search-user”. Hier werden ebenfalls die einzelnen Browser über den “find”-Befehl gesucht. Die konkreten genutzten Suchbefehle für die jeweiligen Browser lauten:

```
1 # Firefox
2 find $tempdir/mount -name places.sqlite 2> /dev/null >> ↩
    $tempdir/temp/browserpath_firefox_temp
3 # Opera
4 find $tempdir/mount -name global_history.dat 2> /dev/null >> ↩
    $tempdir/temp/browserpath_opera_temp
5 find $tempdir/mount -name history.dat 2> /dev/null >> ↩
    $tempdir/temp/browserpath_opera_temp2
6 # Chrome
7 find $tempdir/mount -name History 2> /dev/null >> ↩
    $tempdir/temp/browserpath_chrome_temp
8 # InternetExplorer
9 find $tempdir/mount -name index.dat 2> /dev/null >> ↩
    $tempdir/temp/browserpath_ie_temp
```

Es wird also zunächst für den Firefox nach der Datei “places.sqlite” gesucht, welche seit Version 3.0 die History-Datei für diesen Browser ist. Um die gefundenen Pfade zu verifizieren, wird jeder auf diese Weise gefundene Ordner auf das Vorhandensein der Datei “cookies.sqlite” getestet. Wird diese vorgefunden, handelt es sich um einen gültigen Firefox-Ordner.

Anschließend wird derselbe Ablauf für Opera durchgeführt. Da es mit der Umstellung von Version 9 auf 10 zu einem Umbenennen der History-Datei von “history.dat” auf “global_history.dat” gekommen ist, werden zwei Suchzeilen benötigt. Zum Testen des Ordners wird dieser auf das Vorhandensein der Datei “cookies4.dat” untersucht.

Beim nächsten Webbrowser wird nach Chrome gesucht. Dort nennt sich die History-Datei einfach nur “History”, ohne Endung. Die Validierung des Ordners erfolgt auch hier durch das Prüfen, ob eine Datei vorhanden ist. Diese trägt den Namen “Cookies” und besitzt ebenfalls keine Dateiendung.

Als letzter Browser wird der InternetExplorer gesucht. Dies gestaltet sich umfangreicher, da pro Benutzer mehrere Pfade existieren. Zuerst wird nach der “index.dat”, wie aus dem Befehl ersichtlich, gesucht. Anschließend werden alle gefundenen Einträge abgearbeitet und die letzten fünf Buchstaben des beinhaltenden Ordners gegen folgende Werte verglichen:

- **“okies”:**
  Wird dieses Ergebnis erkannt, so handelt es sich um den Ordner für die Cookies.

- **“y.IE5”:**
  Sollte dieser Ordner gefunden werden, handelt es sich um den Ordner für die eigentliche History.

- **“t.IE5”:**
  Wenn dieser Ordner erkannt wird, handelt es sich um die temporären Inhalts-Daten.

- **“rData”:**
  Wird dieser String erkannt, so handelt es sich um den Ordner mit den Benutzerdaten.

Die genaue Struktur und Bedeutung der einzelnen InternetExplorer-Ordner kann in Kapitel 4.5.1 auf Seite 29 nachgelesen werden. Der Grund, dass exakt fünf Buchstaben verwendet werden, liegt daran, dass dadurch eine eindeutige Zuordnung möglich ist und zusätzliche Zeichen keinen Vorteil bieten würden.

Sind alle Ordner getestet, werden die gültigen Pfade in Dateien im temporären Ordner für die weitere Verwendung gespeichert. Eine Zuordnung der Ordner zu den jeweiligen Browsern erfolgt erst beim Kopiervorgang im Skript “copy-browser”, welches im Kapitel 6.7.11 auf der nächsten Seite zu finden ist.

6.7.10 Skript: temp-prepare

Dieses Skript bereitet den temporären Ordner auf den Kopiervorgang der jeweiligen Browser-Ordner vor. Zunächst wird im temporären Ordner der Unterordner “work” betreten und darin ein Unterordner für jeden Typ der gefundenen Betriebssysteme angelegt. Dies kann also “linux”, “windows_old” oder “windows_new” sein. Sollte nur ein System gefunden werden, so wird ebenfalls nur ein Ordner angelegt.

Anschließend wird die Liste der Benutzer durchgegangen und jeweils ein Unterordner beim entsprechenden Typ mit dem Namen des Users erstellt.

Zuletzt wird in jedem dieser Ordner folgende Struktur angelegt:

```
<Benutzername>/firefox
<Benutzername>/chrome
<Benutzername>/opera
<Benutzername>/ie
<Benutzername>/ie/Cookies
<Benutzername>/ie/History
<Benutzername>/ie/Temporary Internet Files
<Benutzername>/ie/Internet Explorer
<Benutzername>/ie/Internet Explorer/userData
```

Damit ist die Erstellung der Struktur im temporären Ordner abgeschlossen und es kann der eigentliche Kopiervorgang, wie im nächsten Kapitel beschrieben, gestartet werden.

6.7.11 Skript: copy-browser

Durch die Nutzung dieses Skriptes werden alle gefundenen Browser-Ordner in die zuvor angelegte Struktur kopiert. Der Ablauf ist im Folgenden erklärt.

Zunächst werden alle Dateien mit Pfaden zu Browser-Ordnern genutzt und diese zeilenweise eingelesen. Jede dieser Zeilen wird mit allen gefundenen Benutzern verglichen. Beinhaltet ein Pfad den Namen des Users, so wird er diesem fix zugeordnet. Wird kein zugehöriger Benutzer gefunden, wird dieser als “undefined” mit einer laufenden Nummer bezeichnet.

Anschließend werden von jedem Ordner, durch den Aufruf des Skriptes “md5sum-create”, rekursiv MD5-Summen erstellt und diese in eine Datei mit dem Namen “md5db” hinterlegt. Genaueres dazu ist im nächsten Kapitel 6.7.12 auf der nächsten Seite zu finden.

Schließlich wird jeder dieser Ordner mittels “rsync” in die vorbereitete Struktur kopiert. Für die Browser Firefox, Opera und Chrome werden jeweils Verzeichnisse mit einer laufenden Nummer angelegt, da von diesen mehrere Pfade existieren können. Wie die genaue Struktur aufgebaut ist, kann in Kapitel 6.3.2 auf Seite 45 nachgelesen werden.

Da sich aber je nach System der Cache-Ordner nicht im Ordner der History-Datei befindet, muss dieser zusätzlich kopiert werden. Abhängig von System und Browser muss daher Folgendes beachtet und durchgeführt werden:

- **Windows Vista und Windows 7:**
  Hier werden für die Browser Firefox und Opera die Cache-Dateien bei den Anwendungsdaten im Unterordner *Local* untergebracht. Die Dateien mit der History ist hingegen im Unterordner *Roaming* zu finden. Wird in den Optionen des jeweiligen Browsers ein neuer Pfad außerhalb der Anwendungsdaten gesetzt, so befindet sich der Cache automatisch im jeweiligen Verzeichnis. Daher reicht es, den Namen im Pfad zu ersetzen und - sofern dieser so generierte Pfad existiert - mittels “rsync” nach dem Anlegen der jeweiligen MD5-Summen in den bereits kopierten Browser-Ordner zu kopieren.

- **Windows 2000 und Windows XP:**
  Hier verhält es sich ebenso wie im vorher beschriebenen Punkt, allerdings mit einer etwas geänderten Ordnerstruktur für dieselben Browser. So liegt History direkt in den “Anwendungsdaten”, der Cache hingegen im Ordner “Lokale Einstellungen”. Dieser Ordner beinhaltet auch einen Ordner mit dem Namen “Anwendungsdaten”. Somit wird aus dem Ordner “<Benutzerordner>/Anwendungsdaten/Opera” der Pfad “<Benutzerordner>/Lokale Einstellungen/Anwendungsdaten/Opera”. Das Erstellen der MD5-Summen und das Kopieren der Daten erfolgt anschließend auf dieselbe Weise.

- **Linux:**
  Unter Linux muss nur der Browser Chrome angepasst werden. So befinden sich die

Daten zur History im Ordner ".config/google-chrome/", der Cache aber im Ordner ".cache/google-chrome/" im jeweiligen Benutzerordner. Wird der Pfad durch Verändern der Konfiguration verschoben, befindet sich auch hier der Cache wieder automatisch im selben Ordner. Von daher reicht es aus, wenn die Korrektur nur für den Standard durchgeführt wird.

Nachdem alle Ordner inklusive zugehörigem Cache kopiert wurden, werden diese inklusive der enthaltenen Dateien mit den erstellten MD5-Summen mittels "md5-check" verglichen. Fehlerhafte Dateien werden markiert und in den Ordner "err" verschoben. Genaueres dazu befindet sich im zum Skript gehörenden Kapitel 6.7.13.

6.7.12 Skript: md5-generate

Dieses Skript beinhaltet neben dem Auslesen der Konfiguration, nur folgende Zeile:

```
md5deep -r "$2" >> $tempdir/md5db
```

Das bedeutet, dass nur das Programm "md5deep" aufgerufen und die Ausgabe in die Datei "md5db" kopiert wird. Um bei Problemen mit der Ausführung nicht den kompletten Kopiervorgang zu unterbrechen, wurde dieser Befehl ausgelagert. Genaueres dazu ist in Kapitel 6.2 auf Seite 37 zu finden.

6.7.13 Skript: md5-check

Die Aufgabe dieses Skripts besteht darin festzustellen, ob eine Datei im Ordner "work" geändert wurde. Sollte dies der Fall sein, so wird die jeweilige Datei in den Ordner "err" verschoben. Der Abgleich der Dateien mit der Datenbank erfolgt dabei wieder über "md5deep" mit folgendem Befehl:

```
md5deep -r -x $tempdir/md5db "$2" > $tempdir/temp/md5err_tmp
```

Alle Fehler werden dabei inklusive zugehörigem Pfad in die Datei "md5err_tmp" eingetragen. Diese Zeilen werden anschließend genutzt, um die betreffende Datei in den Ordner "err" zu verschieben.

Das Verschieben hat den Zweck, dass die betroffene Datei zwar erhalten bleibt, aber für die restliche Bearbeitung nicht mehr genutzt wird. Somit werden zwar unter Umständen nicht alle besuchten URLs gefunden, es werden allerdings keine eventuell manipulierten Daten weiterverarbeitet.

6.7.14 Skript: search-files

Durch dieses Skript wird das Image auf gelöschte Dateien durchsucht. Dabei wird das gesamte Image, unabhängig vom Dateisystem, durchsucht, wodurch unter anderem auch der Inhalt von SWAP-Partitionen für History-Dateien ausgewertet werden kann.

Die eigentliche Suche findet durch folgenden Befehl mit dem Programm “photorec” statt:

```
/usr/sbin/photorec /log /logname $tempdir/photorec.log /d ↩
    $tempdir/recover/ /cmd $image ↩
    partition_none,options,lowmem,fileopt,everything,disable, ↩
    dat,enable,sqlite,enable,wholespace,search
```

Es wird die Imagedatei nach “*.dat”- und “*.sqlite”-Dateien durchsucht und in den Ordner “recover” kopiert. Dabei werden keine Partitionen berücksichtigt, der Speicherverbrauch möglichst minimiert und ein Log in das temporäre Verzeichnis erstellt. Der Suchvorgang kann abhängig von der Größe des zu durchsuchenden Images und von der Schnelligkeit der Festplatte sehr lange dauern. Tests mit USB-Festplatten (Datendurchsatz um die 25 Megabyte pro Sekunde) haben gezeigt, dass pro Stunde in etwa 70 bis 80 Gigabyte verarbeitet werden können. Die Leistung der CPU kann aufgrund der geringen Datenmenge vernachlässigt werden.

Da das Program “photorec” seine Ausgabe in einen eigenen Framebuffer schreibt, ist im normalen Bash-Log nur eine minimale Ausgabe zu finden. Die Details können in der von der Anwendung erstellen Datei “photorec.log” im temporären Verzeichnis nachgelesen werden.

Nach dem Suchvorgang wird mittels “md5deep” nach bereits vorhandenen Dateien gesucht und diese anschließend gelöscht. Danach werden die neuen Dateien in die MD5-Datenbank eingefügt. Damit ist der Suchvorgang abgeschlossen und die wiederhergestellten Dateien sind für die Verarbeitung bereit.

6.7.15 Skript: umount-image

Dieses Skript löst die Einbindung aller im temporären Verzeichnis gemounteten Datenträger oder Images. Dazu wird die Systemdatei "/etc/mtab" ausgelesen und mittels "grep" nach Zeilen mit dem temporären Ordner gesucht. Die gefundenen Punkte werden anschließend mit folgendem Befehl ausgehängt:

```
sudo umount -f -l ${umountdir[$i]}
```

Mit "-f" wird der Vorgang erzwungen und durch "-l" abgeschlossen, sobald das letzte darauf zugreifende Programm beendet ist. Somit wird sichergestellt, dass nach dem Programmende keine Partition mehr eingehängt ist.

Wurde zum Beispiel eine USB-Festplatte untersucht, könnte diese bereits nach der Ausführung des Skripts vom Rechner abgeschlossen werden. Dies ist möglich, da das restliche Programm sämtliche Informationen aus dem temporären Ordner ausliest und die Datenquelle dadurch nicht mehr benötigt wird. Daher wird ein externen Speichermedium, sofern nicht nach gelöschten Dateien gesucht wird, nur für wenige Minuten benötigt.

6.7.16 Skript: generate-work-db

Mit Hilfe dieses Skripts wird eine Datenbank erstellt, in welcher sämtliche History-Einträge der einzelnen gefundenen und kopierten Browser enthalten sind. Dazu werden auch mehrere Sub-Skripte aufgerufen.

Zunächst wird eine leere Datenbank mit Hilfe des Unterskripts "generate-work-db-create" erstellt. Anschließend wird jeder Systemtyp hintereinander, beginnend mit Linux, bearbeitet. Für jeden Typ wird jeder User der Reihe nach abgearbeitet. Pro Benutzer wird dann die eigentliche Bearbeitung für jeden darin enthaltenen Browser mit einem der folgenden Befehle gestartet:

```
# Firefox
bash $progpath/bin/generate-work-db-ff $tempdir/$configfile ↩
    $systemname $username $browsername
# Opera
```

```
bash $progpath/bin/generate-work-db-opera $tempdir/$configfile ↩
    $systemname $username $browsername
# Chrome
bash $progpath/bin/generate-work-db-chrome $tempdir/$configfile ↩
    $systemname $username $browsername
# InternetExplorer
bash $progpath/bin/generate-work-db-ie $tempdir/$configfile ↩
    $systemname $username $browsername
```

Es wird also ein Sub-Skript pro Browserordner aufgerufen und diesem neben der Konfigurationsdatei, der Systemtyp, der Benutzername und der Browsername zur Untersuchung übergeben.

Nachdem alle zuordenbaren Benutzer abgearbeitet wurden, beginnt die Auswertung der "undefined-<Nummer>"-Ordner. Pro Ordner wird das Skript "generate-work-db-undefined" aufgerufen, um den Browsertyp zu bestimmen und dementsprechend die vorher erwähnten Skripte auszuführen.

Zuletzt analysiert dieses Skript die wiederhergestellten Dateien. Jede Datei wird dabei an das Skript "generate-work-db-recovered" zur genaueren Analyse und darauf basierender Bearbeitung weitergeleitet.

6.7.17 Skript: generate-work-db-create

Dieses Skript dient nur dazu, die Sqlite-Datenbank inklusive darin enthaltener Tabellen anzulegen. Dazu wird eine leere Datei mit Hilfe des Befehls "touch" angelegt. Anschließend wird das Programm "sqlite3" genutzt, um eine gültige Datenbank anzulegen. Zudem werden pro Browser zwei Tabellen angelegt:

1. URL:
   Diese Tabelle beinhaltet die gefundenen URLs inklusive der für den jeweiligen Browser verfügbaren Informationen wie Zeitpunkt, vorherige Webseite oder der Anzahl der Aufrufe.

2. Cookie:
   Diese Tabelle beinhaltet alle gefundenen Cookies für den jeweiligen Browser. Neben

dem Namen und den Werten sind zum Beispiel auch Erstellungs- und Ablaufszeit hinterlegt.

Die einzige Ausnahme bildet, wie bereits in Kapitel 4.3 auf Seite 23 genauer erwähnt, der Browser Opera. Dort exisitiert keine zugehörige Cookies-Tabelle.

Die genauen Möglichkeiten, die aus dem jeweiligen Browser ausgelesen werden können, sind in den Unterkapiteln von Kapitel 4 zu finden.

Im Folgenden ist der Befehl zum Anlegen der URL-Tabelle für den Browser Chrome aufgelistet:

```
sqlite3 $tempdir/workdb.sqlite "create table url_chrome (
    t1key INTEGER PRIMARY KEY,
    systemname TEXT,
    username TEXT,
    browsername TEXT,
    ID int,
    VISIT_TIME long,
    FROM_VISIT int,
    TRANSITION long,
    IS_INDEXED int,
    URL TEXT,
    TITLE TEXT,
    VISIT_COUNT int,
    TYPED_COUNT int,
    LAST_VISIT_TIME long,
    HIDDEN int,
    FAVICON_ID int,
    NAME TEXT,
    PRES_INDEX int,
    TIME_SLOT long,
    SEGMENT_VISIT_COUNT int);"
```

Die Tabelle trägt also den Namen “url_chrome” und beinhaltet 20 Datenfelder mit definierter Art des Inhaltes.

6.7.18 Skript: generate-work-db-ff

Der Zweck dieses Skripts ist das Auslesen des jeweils übergebenen Firefox-Ordners und das Einfügen der gewonnenen Daten in die Datenbank. Der Ablauf für die URL-History ist dabei folgender:

- Zuerst wird die History aus der Datenbank mittels SQL-Befehl ausgelesen und in einer temporären Datei gespeichert. Der dazu verwendete Befehl lautet:

```
sqlite3 "$*" 'SELECT
"moz_places"."url", "moz_places"."title", ↩
   "moz_places"."rev_host", "moz_places"."visit_count", ↩
   "moz_places"."hidden", "moz_places"."typed", ↩
   "moz_places"."frecency", "moz_places"."last_visit_date", ↩
   moz_historyvisits"."id", ↩
   "moz_historyvisits"."from_visit", ↩
   "moz_historyvisits"."visit_date", ↩
   "moz_historyvisits"."visit_type", ↩
   "moz_historyvisits"."session", "moz_favicons"."url", ↩
   "moz_favicons"."mime_type", "moz_favicons"."expiration"
FROM  "moz_places"
LEFT OUTER JOIN "moz_historyvisits" ON ↩
   "moz_places"."id"="moz_historyvisits"."place_id"
LEFT OUTER JOIN "moz_favicons" ON ↩
   "moz_places"."favicon_id"="moz_favicons"."id"' > ↩
   $tempdir/temp/ff_history_dump
```

  Wie aus dem Befehl ersichtlich, werden mehrere Tabellen mittels Join zu einer großen zusammengeführt und anschließend nur die namentlich angegebenen Felder an die Datei "ff_history_dump" im temporären Ordner weitergeleitet.

- Anschließend wird die ausgelesene Datenbank zeilenweise abgearbeitet und die jeweiligen Werte werden entnommen. Beim unerlaubten Zeichen "|" im Titel der Webseite würde es zu Fehlern beim Parsen der SQL-Ausgabe kommen. Daher wurde eine automatische Erkennung und Behebung eingebaut. Leere Integer-Felder

werden anschließend automatisch mit “0” gefüllt. Eine Anpassung des Zeitwertes ist bei Firefox nicht notwendig.

- Die im vorherigen Schritt gewonnenen Werte werden nach jeder Zeile direkt in der neuen Datenbank gesichert. Der Befehl dazu ist folgender:

```
sqlite3 $tempdir/workdb.sqlite "insert into url_ff
(systemname, username, browsername, URL, TITLE, REV_HOST, ↩
   VISIT_COUNT, HIDDEN, TYPED, FRECENCY, LAST_VISIT_DATE, ↩
   VISIT_ID, FROM_VISIT, VISIT_DATE, VISIT_TYPE, SESSION, ↩
   FAV_URL, FAV_MIME, FAV_EXPIRATION)
values
('$systemname', '$username', '$browsername', '$URL', ↩
   '$TITLE', '$REV_HOST', '$VISIT_COUNT', '$HIDDEN', ↩
   '$TYPED', '$FRECENCY', '$LAST_VISIT_DATE', '$VISIT_ID', ↩
   '$FROM_VISIT', '$VISIT_DATE', '$VISIT_TYPE', '$SESSION', ↩
   '$FAV_URL', '$FAV_MIME', '$FAV_EXPIRATION');"
```

Wie zu sehen ist, werden neben den gewonnen Daten auch der Benutzername und der Systemtyp mit in die Tabelle “url_ff” gesichert. Dadurch geht die Zuordnung der URL an den jeweiligen Ursprung nicht verloren.

Ist die komplette alte Datenbank abgearbeitet und sind alle Einträge in die neue übernommen, wird der gleiche Ablauf für die Cookies wiederholt. Es wird also die Datei Cookie-Datenbank wieder mittels SQL-Befehl abgefragt, die Werte werden zeilenweise übernommen und in die Tabelle “cookies_ff” in der neuen Datenbank eingetragen.

6.7.19 Skript: generate-work-db-opera

Dieses Skript dient zum Abarbeiten der Einträge des Browsers Opera. Von den vier möglichen Browsern können hier am wenigsten Informationen gewonnen werden. Näheres zu den Gründen ist in Kapitel 4.3 auf Seite 23 zu finden.

Die Informationsgewinnung basiert auf dem Auslesen der Datei “global_history.dat” oder “history.dat”. Dort sind die Informationen auf vier Zeilen verteilt. Die Information der jeweiligen Zeile ist folgende:

1. Zeile: Hier wird, sofern möglich, der Titel der Webseite angegeben. Ist dies nicht möglich, wird die URL genutzt.
2. Zeile: Hier ist die genaue URL der Webseite zu finden.
3. Zeile: An dieser Stelle wird immer die Zeit des letzten Aufrufes angegeben.
4. Zeile: Hier ist eine Zusatzinformation zum Cache zu finden. Ist keine vorhanden, wird "-1" eingetragen.

Sind vier Zeilen ausgelesen, werden die enthaltenen Informationen zuzüglich Systemtyp und Benutzernamen in die Tabelle "url_opera" eingetragen. Der Befehl dazu entspricht dem im vorherigen Kapitel dargestellten.

Weitere Informationen und Cookies können leider aufgrund des binären Formats und dem Fehlen der näheren Information nicht ausgelesen werden.

6.7.20 Skript: generate-work-db-chrome

Mit Hilfe dieses Skripts können die Werte der jeweiligen Chrome-Datenbank ausgelesen und in die Datenbank des aktuellen Ablaufes eingetragen werden. Das Vorgehen entspricht dabei dem des Browsers Firefox, welches in Kapitel 6.7.18 auf Seite 80 beschrieben wurde.

Es wird zunächst die vorhandene History-Datenbank mit einem SQL-Befehl, welcher auch Joins enthält, ausgelesen. Anschließend werden die Werte zeilenweise abgearbeitet und direkt in die neue Datenbank eingetragen. Dasselbe gilt auch für die Cookies.

Beim zeilenweisen Einlesen der Werte ist ebenfalls eine Erkennung und Behebung für das unerlaubte Zeichen "|" integriert. Dies gilt auch für das automatische Füllen der leeren Integer-Werte mit "0".

6.7.21 Skript: generate-work-db-ie

Dieses Skript steuert das Programm "pasco" an und trägt die zurückgegebenen Informationen in der Datenbank ein. Der genaue Ablauf ist folgender:

- Zuerst wird die übergebene "index.dat" durch das Unterskript "generate-work-db-ie-pasco" ausgelesen.

- Die gewonnenen Informationen werden zeilenweise analysiert. Die möglichen auslesbaren Informationen sind in Kapitel 4.5.2 auf Seite 30 beschrieben.
- Pro Zeile wird eine automatische Korrektur für leere Felder vorgenommen. Dies ist notwendig, da die Ausgabe von "pasco" dies nicht berücksichtigt.
- Anschließend wird getestet, ob der angegebene Cache noch existiert. Die entsprechende Information wird ebenfalls gespeichert.
- Schließlich wird die ausgelesene Zeit noch analysiert und leere Felder werden auf "0" korrigiert.
- Ist dies erfolgt, wird die jeweilige Zeile in die Datenbank eingetragen. Der dazu notwendige SQL-Befehl entspricht denen der bereits besprochenen Browser.

Sollte sich in der "index.dat" mindestens ein URL-Eintrag befinden, der mit "Cookie:" beginnt, wird sofort mit dem Auslesen der Cookies über das Skript "generate-work-db-ie-cookie" gestartet und die Verarbeitung erst nach dessen Beendigung fortgesetzt.

6.7.22 Skript: generate-work-db-ie-pasco

Dieses Skript hat nur die Funktion, das Programm "pasco" aufzurufen und die Ausgabe in eine Datei weiterzuleiten. Dies ist notwendig, da mehrere Tests gezeigt haben, dass das Programm bei manchen "index.dat"-Dateien abstürzen kann. Ist dies der Fall, können die bis dorthin ausgelesenen Daten noch verarbeitet werden und der Programmablauf wird nicht abgebrochen. Informationen darüber, warum dies in Bash notwendig ist, können in Kapitel 6.2 auf Seite 37 nachgelesen werden.

Bei den Tests gab es von fünf Windows-Systemen genau eine "index.dat", bei welcher dieser Schritt notwendig war. Bei dieser Datei handelte es sich um eine InternetExplorer 6.0 History-Datei von WindowsXP mit ServicePack 2. Der genaue Fehler war folgender:

```
1 *** glibc detected *** /usr/sbin/pasco: double free or corruption ↩
    (!prev): 0x093354b0 ***
2 ======= Backtrace: =========
3 /lib/i686/cmov/libc.so.6(+0x6b321)[0xb761f321]
4 /lib/i686/cmov/libc.so.6(+0x6cb78)[0xb7620b78]
```

```
5 /lib/i686/cmov/libc.so.6(cfree+0x6d)[0xb7623c5d]
6 /usr/sbin/pasco[0x80491d3]
7 /usr/sbin/pasco[0x8049684]
8 /lib/i686/cmov/libc.so.6(__libc_start_main+0xe6)[0xb75cac76]
9 /usr/sbin/pasco[0x8048681]
10 ======== Memory map: ========
11 08048000-0804a000 r-xp 00000000 08:01 3440996    /usr/sbin/pasco
12 0804a000-0804b000 rw-p 00001000 08:01 3440996    /usr/sbin/pasco
13 09335000-09356000 rw-p 00000000 00:00 0          [heap]
14 b7400000-b7421000 rw-p 00000000 00:00 0
15 b7421000-b7500000 ---p 00000000 00:00 0
16 b7575000-b7592000 r-xp 00000000 08:01 229379     /lib/libgcc_s.so.1
17 b7592000-b7593000 rw-p 0001c000 08:01 229379     /lib/libgcc_s.so.1
18 b75b2000-b75b4000 rw-p 00000000 00:00 0
19 b75b4000-b76f4000 r-xp 00000000 08:01 246281     ↩
     /lib/i686/cmov/libc-2.11.2.so
20 b76f4000-b76f5000 ---p 00140000 08:01 246281     ↩
     /lib/i686/cmov/libc-2.11.2.so
21 b76f5000-b76f7000 r--p 00140000 08:01 246281     ↩
     /lib/i686/cmov/libc-2.11.2.so
22 b76f7000-b76f8000 rw-p 00142000 08:01 246281     ↩
     /lib/i686/cmov/libc-2.11.2.so
23 b76f8000-b76fb000 rw-p 00000000 00:00 0
24 b76fb000-b771f000 r-xp 00000000 08:01 246265     ↩
     /lib/i686/cmov/libm-2.11.2.so
25 b771f000-b7720000 r--p 00023000 08:01 246265     ↩
     /lib/i686/cmov/libm-2.11.2.so
26 b7720000-b7721000 rw-p 00024000 08:01 246265     ↩
     /lib/i686/cmov/libm-2.11.2.so
27 b773f000-b7742000 rw-p 00000000 00:00 0
28 b7742000-b7743000 r-xp 00000000 00:00 0          [vdso]
29 b7743000-b775e000 r-xp 00000000 08:01 229401     /lib/ld-2.11.2.so
30 b775e000-b775f000 r--p 0001a000 08:01 229401     /lib/ld-2.11.2.so
31 b775f000-b7760000 rw-p 0001b000 08:01 229401     /lib/ld-2.11.2.so
32 bfc86000-bfc9b000 rw-p 00000000 00:00 0          [stack]
33 /home/user/Masterarbeit/msc/bin/generate-work-db-ie-pasco: Zeile ↩
```

```
43: 13882 Abgebrochen              /usr/sbin/pasco "$2" > ↩
$tempdir/$tempfile 2> /dev/null
```

Wie zu erkennen ist, handelt es sich um einen Pointer-Fehler direkt im Programm, welcher bisher nicht behoben wurde. Eine entsprechende Meldung per Mail an den Entwickler wurde ignoriert, weshalb nichts gegen dieses Problem unternommen werden konnte. Mangels freier Alternativen zu "pasco" konnte das Programm auch nicht ersetzt werden.

Sollte ein ähnlicher Fehler während der Bearbeitung auftreten, empfiehlt es sich, die jeweilige Datei händisch mit einem anderen Programm zu untersuchen. Eine Auflistung möglicher Programme ist in Kapitel 3.5 auf Seite 9 zu finden.

6.7.23 Skript: generate-work-db-ie-cookie

Dieses Skript liest alle Cookies vom übergebenen Ordner aus und speichert die Information in die Datenbank. Das Vorgehen ist dabei identisch mit dem des Skripts "generate-work-db-ie-pasco", nur dass statt "pasco" das Programm "galleta" aufgerufen wird.

Der Inhalt der Cookies wird mittels des erwähnten Programms ausgelesen und die gefundenen Daten werden zeilenweise verarbeitet. Leere Zeitfelder werden hier ebenfalls automatisch mit "0" aufgefüllt, um spätere Fehler zu vermeiden. Die jeweilige Zeile wird abschließend mittels SQL-Befehl in die Datenbank eingetragen.

6.7.24 Skript: generate-work-db-ie-galleta

Wie auch beim Skript "generate-work-db-ie-pasco" wird hier nur "galleta" aufgerufen und die Ausgabe in eine Datei gesichert. Es traten zwar bei den Tests keine Fehler mit dem Programm auf, da es jedoch von derselben Quelle stammt, handelt es sich um eine reine Vorsichtsmaßnahme.

6.7.25 Skript: generate-work-db-undefined

Dieses Skript startet für den jeweiligen Browserordner das zugehörige Skript. Bei der Planung des Programms waren weitere Funktionen vorgesehen, was sich jedoch im Zuge der Implementierung als unnötig herausgestellt hat. Daher ist dies die einzige Funktion. Eine Zusammenführung mit dem Skript "generate-work-db" wäre zwar möglich, im Hinblick auf einfachere eventuelle Erweiterungen wurde jedoch darauf verzichtet.

Aufgerufen kann eines der folgenden Skripte werden:

- Skript: generate-work-db-ff
- Skript: generate-work-db-opera
- Skript: generate-work-db-chrome
- Skript: generate-work-db-ie

Die laufende Nummer der unzuordenbaren Ordner wird ebenfalls in die Datenbank eingetragen. Dadurch ist eine Unterscheidung weiterhin möglich und den jeweiligen Dateien im temporären Ordner zuordenbar.

6.7.26 Skript: generate-work-db-recovered

Dieses Skript analysiert den übergebenen Ordner mit den wiederhergestellten Dateien und startet das entsprechende Skript für den jeweils erkannten Browser. Handelt es sich bei der gefundenen Datei um eine SQLite-Datenbank, so wird diese gegen den Browser Firefox und Chrome getestet, andernfalls gegen Opera oder InternetExplorer. Der Testvorgang für den jeweiligen Browser ist folgender:

1. Es wird getestet, ob es sich um eine Firefox-History-Datei handelt, indem einfach ein “sqlite3” mit einem SQL-Befehl genutzt wird. Sollte es sich um eine entsprechende Datenbank handeln, werden Daten zurückübergeben. Andernfalls wird die Fehlermeldung erkannt.

2. Für den Browser Chrome ist das Vorgehen bis auf den genutzten SQL-Befehl identisch.

3. Da der Browser Opera alle Informationen im Klartext auflistet, kann gezielt nach enthaltenen URLs gesucht werden. Wird eine gültige Zeile gefunden, so handelt es sich mit ziemlicher Sicherheit um diesen Browser. Eine Alternative würde ohnehin nicht in Frage kommen, da keiner der anderen History-Dateien aus Klartext besteht.

4. Da die gültigen “index.dat” Dateien mit dem Wort “Client” beginnen, wird die Datei genau auf dies hin untersucht. Sollte es sich zufällig um eine reine Textdatei mit demselben Wort am Anfang handeln, wird die Auswertung spätestens beim

Aufruf von “pasco” unterbrochen. Es ist aber auf jeden Fall sichergestellt, dass keine falsche Zeile in die Datenbank eingetragen wird.

Sollte es sich bei der gefundenen Datei um eine Browser-History-Datei handeln, wird das jeweilige Skript für die Auswertung gleich wie bei dem Skript “generate-work-db-undefined” gestartet.

Bei dieser Methode können keine Cookies ausgewertet werden, weil die zugehörige Datei zwar wiederherstellbar aber nicht eindeutig zuordenbar ist. Dasselbe gilt auch für die Verarbeitung des Cache.

6.7.27 Skript: md5-check-second

Dieses Skript testet erneut die MD5-Summen im Ordner “work”. Dabei wird erneut das Skript “md5-check” aufgerufen. Informationen dazu können in Kapitel 6.7.13 auf Seite 75 nachgelesen werden. Das Skript hat nur den Zweck, eine saubere Textausgabe auf der Konsole zu erzeugen.

6.7.28 Skript: generate-check-db

Mit Hilfe dieses Skripts werden alle für die Auswertung der erstellten Datenbank notwendigen Unterskripte aufgerufen. Dadurch wird eine neue Datenbank erstellt, in der die gefundenen URLs aller Browser kombiniert und getestet werden. Der Ablauf ist folgender:

1. Erstellen der neuen Datenbank mit Hilfe des Skripts “generate-check-db-create”.
2. Kopieren aller Cookies in die neue Datenbank durch das Skript “generate-check-db-cookie”.
3. Verarbeiten aller Firefox-Einträge durch das Skript “generate-check-db-ff”. Die Daten werden dabei aus der alten Datenbank, wie auch bei den nachfolgenden drei Punkten, entnommen.
4. Verarbeiten aller Opera-Einträge durch das Skript “generate-check-db-opera”.
5. Verarbeiten der Einträge von Chrome durch “generate-check-db-chrome”.
6. Verarbeiten der Einträge des InternetExplorers durch “generate-check-db-ie”.

7. Abschließend werden die Einträge der neue Datenbank durch das Skript “generate-check-db-update” aktualisiert und die Browsersitzungen finalisiert.

Die Aufgabe dieses Skripts ist daher ähnlich wie die von “main” oder “generate-work-db”. Es werden also keine Daten direkt verarbeitet, sondern nur weitere Skripte für den jeweiligen Zweck aufgerufen.

6.7.29 Skript: generate-check-db-create

Dieses Skript erstellt eine neue Datenbank, in welcher alle gefundenen Einträge kombiniert und ausgewertet werden. In dieser Datenbank werden folgende zwei Tabellen erstellt:

- **“finaldb_url”:**
  Hier werden später alle gefundenen URLs inklusive der einzelnen Testergebnisse eingetragen.

- **“finaldb_cookie”:**
  In diese Tabelle werden alle Cookies kopiert. Zusätzlich werden alle bei den Tests gefundenen Cookies im Laufe der Verarbeitung markiert.

Das Anlegen der Tabelle mittels SQL-Befehl funktioniert ebenso wie im Skript “generate-work-db-create”, welches im Kapitel 6.7.17 auf Seite 78 beschrieben ist. Sämtliche Felder aus der Tabelle “finaldb_url” werden später auch in die XML-Dateien exportiert. Folgende Felder werden dabei angelegt:

- **t1key:**
  Hier wird eine laufende Nummer für alle Einträge erstellt. Dies ist der Master-Key für die Tabelle und der einzige Punkt, der später nicht in die XML-Datei exportiert wird.

- **key:**
  Für jeden Browser gibt es pro Eintrag eine laufende Nummer, die mit eins beginnt.

- **systemname:**
  In dieser Tabelle finden sich die Einträge zum jeweiligen Typ des Herkunftssystems. Möglich ist dabei “windows-old”, “windows-new” oder “linux”.

- **username:**
  Hier ist der jeweilige Benutzername zu finden. Bei unzuordenbaren Daten wird "unknown" mit einer laufenden Nummer eingetragen.
- **browsername:**
  Hier wird der Name des Browsers eingetragen, in welchem die aktuelle Zeile gefunden wurde. Mögliche Werte sind "firefox", "opera", "chrome" oder "ie".
- **ID:**
  Diese ID wird aus den History-Daten des jeweiligen Browsers entnommen. Die Zahlen bei "FROM_Visit" beziehen sich auf diese ID.
- **TYPE:**
  Hier wird der Typ des Eintrages angegeben. Möglich ist neben "http", "https", "cookie", "visited" oder "place". Die Daten werde dabei dem jeweiligen Browser entnommen.
- **URL:**
  In dieses Feld wird die eigentliche Adresszeile eingetragen.
- **TITLE:**
  Hier wird der Name der Webseite für die aktuelle URL eingetragen.
- **LAST_VISIT_TIME:**
  Der Zeitpunkt des letzten Besuchs wird in diesem Feld hinterlegt.
- **VISIT_COUNT:**
  In diesem Feld ist zu finden, wie oft die entsprechende URL aufgerufen wurde.
- **TYPED_COUNT:**
  Hier wird die Anzahl, wie oft eine URL direkt eingegeben wurde, gespeichert.
- **FROM_VISIT:**
  In dieses Feld wird die ID der vorherigen URL eingetragen. Für die XML-Datei wird später direkt die jeweilige URL eingetragen.
- **HIDDEN:**
  Hier wird hinterlegt, ob es sich bei der Zeile um eine versteckte, also im Hintergrund geladene URL, handelt.

- **TEST_SESSION_NUMBER:**
  In dieses Feld wird die berechnete Sitzungsnummer eingetragen, welche pro Browser mit eins beginnt.
- **TEST_SESSION_TIME_START:**
  Hier wird die zur Sitzungsnummer gehörende Startzeit eingetragen.
- **TEST_SESSION_TIME_STOP:**
  Neben der Startzeit wird für die jeweilige Sitzung auch die Endzeit hinterlegt.
- **TEST_CACHE_EXIST:**
  Hier wird eingetragen, ob sich die aktuelle URL in einer der Cache-Dateien finden lässt.
- **TEST_CACHE_SAME:**
  Hier wird hinterlegt, ob der Hashwert der heruntergeladenen URL mit einem aus dem Cache-Ordner übereinstimmt. Bei HTML-Seiten wird dies aufgrund der Möglichkeit von dynamischen Inhalten fast nie der Fall sein. Bei Downloads hingegen funktioniert diese Auswertung problemlos.
- **TEST_URL_EXIST:**
  In dieses Feld wird eingetragen, ob die gefundene URL existiert und erreichbar ist.
- **TEST_URL_TITLE_SAME:**
  Hier wird das Ergebnis des Titelvergleichs zwischen gefundenen Daten und heruntergeladener Webseite eingetragen.
- **TEST_COOKIES_SET:**
  Die Anzahl der gesetzten Cookies nach dem Herunterladen der Webseite wird in dieses Feld gespeichert.
- **TEST_COOKIES_EXIST:**
  In dieses Feld wird eingetragen, wie viele zur URL gehörenden Cookies im jeweiligen Browserordner gefunden wurden.
- **TEST_COOKIES_TIME_OK:**
  Hier wird eingetragen, bei wie vielen Cookies der Zeitstempel zwischen den gefundenen und heruntergeladenen ähnlich ist. Dies bezieht sich vor allem auf die Ablaufszeit und die damit verbundene Gültigkeitsdauer.

Die Felder bei den Cookies sind mit denen der “work-db”-Datenbank bis auf das zusätzliche Feld “COOKIE_IS_USED” identisch. In dieses Feld wird eingetragen, ob ein Cookie während der Verarbeitung ähnlich wie bei einem der heruntergeladenen bewertet wurde.

6.7.30 Skript: generate-check-db-cookie

In diesem Skript werden im Wesentlichen nacheinander vier SQL-Abfragen in der alten “work-db”-Datenbank durchgeführt und die ausgelesenen Daten direkt in die Tabelle “fialdb_cookie” in der neuen Datenbank eingetragen.

Zusätzlich werden alle gefundenen Zeiten für den Browser Chrome in die Unix-Time umgerechnet. Warum dies notwendig ist, kann bei der Beschreibung des Skripts “generate-check-db-chrome” in Kapitel 6.7.35 auf Seite 95 nachgelesen werden.

6.7.31 Skript: generate-check-db-insert

Dieses Skript dient dazu, Einträge in der Datenbank “check-db” und dort in der Tabelle “finaldb_url” zu erstellen. Folgende vier Skripte liefern dazu Daten:

- Skript: generate-check-db-ff
- Skript: generate-check-db-opera
- Skript: generate-check-db-chrome
- Skript: generate-check-db-ie

Die jeweiligen einzutragenden Daten werden dabei durch eine temporäre Datei übergeben. Der Vorgang ist identisch mit dem der Konfigurationsübergabe oder dem Auslesen der Sprachzeile.

Nach dem Einlesen aller Werte werden alle Felder zur Sicherheit erneut geprüft. So werden auch eventuelle leere Integer-Felder automatisch mit “0” gefüllt.

Anschließend wird die Zeile mittels SQL-Eintrag in die Datenbank eingetragen. Welche Informationen dabei eingetragen werden, sind in Kapitel 6.7.29 auf Seite 88 zu finden.

6.7.32 Skript: generate-check-db-url

Dieses Skript dient zum Herunterladen einer übergebenen URL. Der Aufruf erfolgt über dieselben Skripte wie bei "generate-check-db-insert". Übergeben wird dabei der Browsertyp und - sofern in der History angegeben - der Referer. Mit diesen Daten wird ein "wget"-Aufruf generiert und es werden die Daten der Adresse inklusive Cookies heruntergeladen. Folgende Referer werden beim jeweiligen Browsertyp verwendet:

```
# Firefox:
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.6) ↩
    Gecko/20070725 Firefox/2.0.0.6
# Opera:
Opera/9.80 (X11; Linux i686; U; de) Presto/2.6.30 Version/10.62
# Chrome:
Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.7 (KHTML, ↩
    like Gecko) Chrome/7.0.517.0 Safari/534.7
# InternetExplorer:
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
```

Der Referer wird beim Abrufen der Webseite benötigt, da viele Webserver diese Kennung auslesen und dementsprechend unterschiedliche Webseiten übertragen. Der eigentliche Aufruf von "wget" ist anschließend folgender:

```
# download the given url
wget -q --cookies=on --keep-session-cookies \
  --save-cookies="$tempdir/wget/$browser/$number/cookies" \
  "$url" -O "$tempdir/wget/$browser/$number/url" \
  --referer="$referer" \
  --user-agent="$agent" \
  --header="Accept:text/xml, application/xml, ↩
     application/xhtml+xml, text/html;q=0.9,text/plain;q=0.8, ↩
     image/png, */*;q=0.5" \
  --header="Accept-Language: en-us,en;q=0.5" \
  --header="Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7" \
  --header="Keep-Alive: 300" "$@"
```

Die Ausgabe wird dabei in einem pro URL erstellten Ordner unter den Namen “url” und “cookies” für die weitere Verarbeitung gespeichert. In der Datenbank im Feld “key”, das sich in der Tabelle “finaldb_url” befindet, ist dieselbe laufende Nummer ebenfalls zu finden. Dadurch kann später eine Verbindung zwischen XML-Dateien und den jeweiligen Downloads hergestellt werden.

6.7.33 Skript: generate-check-db-ff

Mit diesem Skript werden alle Einträge von der alten Datenbank “work-db” in die neue “check-db” überführt. Dabei werden mehrere Anpassungen und Tests durchgeführt. Zunächst wird die alte Datenbank mittels SQL-Befehl, welcher ähnlich dem in Kapitel 6.7.18 auf Seite 80 dargestellt ist, ausgelesen. Anschließend werden diese Daten zeilenweise ausgelesen und folgende Punkte in der angegebenen Reihenfolge durchgeführt:

1. Auslesen der Werte in lokale Variablen. Dabei werden alle Zeitpunkte durch eine Million dividiert, um eine korrekte UNIX-Zeit zu bekommen. Dies ist notwendig, da die Aufzeichnung in Nanosekunden - und nicht wie vom Standard vorgesehen in Sekunden - durchgeführt wird.

2. Beim nächsten Schritt wird die übergebene Zeit ausgewertet und es werden die jeweiligen Sitzungen pro Browser, pro User und pro System erstellt. Einträge, deren Zeitpunkt weniger als 3600 Sekunden auseinander liegen, werden als eine Sitzung gewertet. Die Start- und Endzeit ist dabei die des jeweils ersten und letzten Eintrags. Da während der Auswertung die Endzeit erst ermittelt wird, trägt das Skript “generate-check-db-update” diese später nach. Damit dies möglich ist, werden die jeweiligen Werte in die temporäre Datei “last_visit_time” eingetragen.

3. Anschließend werden die Testvariablen zurückgesetzt, wodurch alles für neue Tests vorbereitet wird. Dazu gehört auch das Anlegen eines Ordners mit laufender Nummer für den Download der angegebenen URL.

4. Danach wird im ersten Testverfahren die aktuelle URL mittels “generate-check-db-url” heruntergeladen. Ist dies möglich, so existiert die jeweilige URL noch und der Test ist positiv.

5. Beim zweiten Test wird der Titel der Webseite in der soeben heruntergeladenen Datei gesucht. Wird ein identischer Text gefunden, gilt dieser als erfolgreich.

6. Im dritten Test wird geprüft, ob ein Eintrag für die aktuelle URL in einer der Cache-Dateien existiert. Dabei werden die Dateien "_CACHE_001_" bis "_CACHE_003_" auf die genaue URL hin durchsucht. Wird ein Eintrag gefunden, so existieren Cache-Daten.

7. Beim vierten Test wird geprüft, ob irgendeine Datei im Cache-Ordner denselben Hashwert besitzt wie die heruntergeladene. Wenn eine Übereinstimmung gefunden wurde, wird der Test als erfolgreich gewertet. Bei HTML-Dateien ist allerdings eine Übereinstimmung sehr unwahrscheinlich, da diese aus zu vielen dynamischen Inhalten bestehen. Es wird zwar durch die Angabe des Browserident und des Referer versucht den jeweiligen Typ zu imitieren, doch bereits eine einfache Zeitanzeige verändert den Hashwert. Dieser Test ist daher vor allem für Bilder, Dokumente und andere Downloads geeignet.

8. Beim nächsten (fünften) Test wird die heruntergeladene Datei "cookies" auf die Anzahl der darin enthaltenen Cookies analysiert und diese Anzahl in eine lokale Variable eingetragen.

9. Der sechste Test besteht darin, die jeweiligen Cookies in der "finaldb_cookies" zu suchen. Benutzer, Browser und System werden dabei natürlich auch berücksichtigt. Die Anzahl der Funde wird als Ergebnis des Tests verwendet. Zusätzlich werden die gefundenen Cookies in der Tabelle markiert.

10. Beim siebten und letzten Test wird die Ablaufzeit der heruntergeladenen Cookies mit denen der gefundenen verglichen. Dabei wird der Zeitunterschied zwischen der jeweiligen Erstellung ermittelt und in den Vergleich miteinbezogen. Ist der Ablaufzeitpunkt bis auf 3600 Sekunden ähnlich, wird dies als gültig gewertet. Die Summe der als gültig erkannten Cookies wird als Ergebnis in die Datenbank eingetragen.

11. Im nächsten Schritt werden alle Daten inklusive der Testergebnisse in eine lokale temporäre Datei gespeichert. Anschließend wird das Skript "generate-check-db-insert" zum Eintragen in die Datenbank aufgerufen.

Sind alle Zeilen abgearbeitet, ist die Aufgabe dieses Skripts abgeschlossen. Die nachfolgenden drei Kapitel nutzen dieselben Tests, weshalb diese dort nicht erneut beschrieben werden.

6.7.34 Skript: generate-check-db-opera

Dieses Skript dient zum Übertragen und Testen der Daten zwischen den beiden Datenbanken. Der Ablauf entspricht dabei dem des Skripts "generate-check-db-ff", welches in Kapitel 6.7.18 auf Seite 80 beschrieben wurde. Auch hier werden die History-Einträge zeilenweise abgearbeitet.

Zunächst werden die Daten jeder Zeile in lokalen Variablen gespeichert, wobei allerdings keine Anpassung der Zeit notwendig ist. Anschließend wird die Sitzung berechnet und die Tests werden durchgeführt. Aufgrund des bereits mehrfach erwähnten Problems, dass die Cookies aufgrund des binären Formats nicht auslesbar sind, gibt es bei den Tests sechs und sieben immer nur "0" als Ergebnis.

Der Ablauf der einzelnen Tests ist dabei mit denen des Skripts "generate-check-db-ff" identisch, weshalb dies hier nicht wiederholt wird.

6.7.35 Skript: generate-check-db-chrome

Auch dieses Skript dient zum Überführen und Testen der Einträge zwischen den beiden Datenbanken. Der Ablauf ist ebenfalls identisch mit dem des Skripts "generate-check-db-ff", welches in Kapitel 6.7.18 auf Seite 80 beschrieben wurde.

Es wird auch hier zunächst die alte Datenbank ausgelesen, anschließend werden die Ergebnisse zeilenweise abgearbeitet. Die Daten jeder Zeile werden in einer lokalen Variablen gespeichert und anschließend wird die Sitzung berechnet. Zusätzlich werden noch die beschriebenen sieben Tests durchgeführt. Die gewonnenen Informationen werden dann über das Skript "generate-check-db-insert" in die Datenbank eingetragen.

6.7.36 Skript: generate-check-db-ie

Mit Hilfe dieses Skripts werden - gleich wie bei den drei zuvor beschriebenen - die Daten zwischen der alten und neuen Datenbank kopiert und getestet. Der genaue Ablauf entspricht ebenfalls dem des Skripts "generate-work-db-ff" (Kapitel 6.7.18 auf Seite 80).

Es werden wiederum im Großen und Ganzen die Einträge der alten Datenbank ausgelesen und zeilenweise abgearbeitet. Nach dem Eintrag der Daten der jeweiligen Zeile in eine lokale Variable werden die Sitzungen berechnet und die Tests durchgeführt. Das Ergebnis wird abschließend wieder über das Skript "generate-work-db-insert" in die Datenbank eingetragen.

Bei den Tests gibt es zwei Besonderheiten:

1. In der "index.dat" wird kein Titel, sondern der Header angegeben. Die heruntergeladene Seite wird daher auf das Vorhandensein derselben Angaben vergleichen und entsprechend bewertet.

2. Das Testen der Existenz des Cache konnte bereits im Skript "generate-work-db-ie", welches in Kapitel 6.7.21 auf Seite 82 beschrieben wurde, durchgeführt werden. Daher muss nur mehr das entsprechende Ergebnis genutzt werden.

Wurden alle Daten überspielt und getestet, ist die Aufgabe dieses Skripts abgeschlossen.

6.7.37 Skript: generate-check-db-update

Dieses Skript wird als letzter Schritt des Bereiches zur Erstellung der zweiten Datenbank ausgeführt. Dabei wird die Sitzungszeit vervollständigt und die jeweilige Endzeit zu jedem Eintrag hinzugefügt. Ebenso wird die temporäre Datei "last_visit_time", in welcher alle Sitzungen inklusive Zeiten, Benutzername, Browsername und Systemtyp hinterlegt sind, zeilenweise ausgelesen und ein darauf basierender SQL-Befehl ausgeführt. Der genaue Befehl lautet:

```
sqlite3 $tempdir/checkdb.sqlite "update finaldb_url set ↩
   TEST_SESSION_TIME_STOP='$session_time_end' where ↩
   (TEST_SESSION_NUMBER='$session_nummer' and ↩
   systemname='$session_system' and ↩
   browsername='$session_browser' and username='$session_user')"
```

Es werden also pro Aufruf alle Felder mit der Endzeit gefüllt, bei denen die Sitzungsnummer, der Systemtyp, der Browsername und der Systemname übereinstimmen.

6.7.38 Skript: save-xml

Mit Hilfe dieses Skripts werden die Daten der erstellten "workdb"-Datenbank in eine oder mehrere XML-Dateien exportiert. Der genaue Importtyp hängt von der zum Programmstart angegebenen Option ab. Möglich sind folgende Optionen:

- **"all":**
  Mit dieser Option werden alle möglichen Ausgaben ("user", "browser", "system" und "one") erstellt.

- **"single":**
  Mit dieser Option werden alle einzelnen Ausgaben ("user", "browser" und "system") erstellt.

- **"user":**
  Dadurch wird eine XML mit folgender Hierarchie erstellt:

```
<whft>
  <user>
    <username>XYZ</username>
    <system>
      <systemname>XYZ</systemname>
      <browser>
        <browsername>XYZ</browsername>
        <entry>
          <id>XYZ</id>
          <type>XYZ</type>
          <url>XYZ</url>
          <title>XYZe</title>
          <from_visit>XYZ</from_visit>
          <last_visit_time>XYZ</last_visit_time>
          <visit_count>XYZ</visit_count>
          <typed_count>XYZ</typed_count>
          <hidden>XYZ</hidden>
          <test_session_number>XYZ</test_session_number>
          <test_session_time_start>XZ</test_session_time_start>
```

```
            <test_session_time_stop>XYZ</test_session_time_stop>
            <test_cache_exist>XYZ</test_cache_exist>
            <test_cache_same>XYZ</test_cache_same>
            <test_url_exist>XYZ</test_url_exist>
            <test_url_title_same>XYZ</test_url_title_same>
            <test_cookies_set>XYZ</test_cookies_set>
            <test_cookies_exist>XYZ</test_cookies_exist>
            <test_cookies_time_ok>XYZ</test_cookies_time_ok>
          </entry>
        </browser>
      </system>
    </user>
</whft>
```

Die Zeichenfolge "XYZ"gibt dabei diejenige Position an, an welcher später Daten zu finden sind. Das heißt, es existiert pro Benutzer ein Unterpunkt für jeden Systemtyp. Zusätzlich enthält jeder Systemtyp Einträge für jeden Browser mit zugehörigen Unterpunkten. Zu beachten ist dabei, dass leere Kombinationen nicht exportiert werden.

- **“browser”:**
  Der Aufbau ist ähnlich dem des zuvor erwähnten. Hier existiert für jeden Browser ein Eintrag für jeden Systemtyp, welcher wiederum einen Eintrag für jeden vorhandenen Benutzer enthält. Die einzelnen Daten sind bei den jeweiligen Benutzern zu finden.

- **“system”:**
  Auch dieser Aufbau ist ähnlich den beiden bereits beschriebenen. Hier beginnt die Hierarchie mit dem Systemtyp. Darin enthalten sind die zugehörigen Benutzer, welche die einzelnen Browser beinhalten. Die Daten selbst sind bei den jeweiligen Browsern zu finden.

- **“one”:**
  Hier werden alle Daten direkt ausgegeben. Die einfache Hierarchie ist dabei folgende:

```
<whft>
  <entry>
    <id>XYZ</id>
    <systemname>XYZ</systemname>
    <username>XYZ</username>
    <browsername>XYZ</browsername>
    <type>XYZ</type>
    <url>XYZ</url>
    <title>XYZ</title>
    <from_visit>XYZ</from_visit>
    <last_visit_time>X</last_visit_time>
    <visit_count>XYZ</visit_count>
    <typed_count>XYZ</typed_count>
    <hidden>XYZ</hidden>
    <test_session_number>XYZ</test_session_number>
    <test_session_time_start>XYZ</test_session_time_start>
    <test_session_time_stop>XYZ</test_session_time_stop>
    <test_cache_exist>XYZ</test_cache_exist>
    <test_cache_same>XYZ</test_cache_same>
    <test_url_exist>XYZ</test_url_exist>
    <test_url_title_same>XYZ</test_url_title_same>
    <test_cookies_set>XYZ</test_cookies_set>
    <test_cookies_exist>XYZ</test_cookies_exist>
    <test_cookies_time_ok>XYZ</test_cookies_time_ok>
  </entry>
</whft>
```

Wie zu erkennen ist, existieren nur die einzelnen Einträge. Bei jedem Eintrag sind der zugehörige Benutzername, der Browsername und der Systemtyp aufgelistet.

Damit es sich um eine gültige XML-Datei handelt, wurde in den Header auch die Beschreibung mit aufgenommen. Die zum aufgeführten Beispiel “user” gehörende Beschreibung ist folgende:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!ELEMENT whft (user)>
<!ELEMENT user (username,system)>
<!ELEMENT username (#PCDATA)>
<!ELEMENT system (systemname,browser)>
<!ELEMENT systemname (#PCDATA)>
<!ELEMENT browser (browsername,entry)>
<!ELEMENT browsername (#PCDATA)>
<!ELEMENT entry (id, type, url, title, from_visit, ↩
    last_visit_time, visit_count, typed_count, hidden, ↩
    test_session_number, test_session_time_start, ↩
    test_session_time_stop, test_cache_exist, test_cache_same, ↩
    test_url_exist, test_url_title_same, test_cookies_set, ↩
    test_cookies_exist, test_cookies_time_ok)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT browsername (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT from_visit (#PCDATA)>
<!ELEMENT last_visit_time (#PCDATA)>
<!ELEMENT visit_count (#PCDATA)>
<!ELEMENT typed_count (#PCDATA)>
<!ELEMENT hidden (#PCDATA)>
<!ELEMENT test_session_number (#PCDATA)>
<!ELEMENT test_session_time_start (#PCDATA)>
<!ELEMENT test_session_time_stop (#PCDATA)>
<!ELEMENT test_cache_exist (#PCDATA)>
<!ELEMENT test_cache_same (#PCDATA)>
<!ELEMENT test_url_exist (#PCDATA)>
<!ELEMENT test_url_title_same (#PCDATA)>
<!ELEMENT test_cookies_set (#PCDATA)>
<!ELEMENT test_cookies_exist (#PCDATA)>
<!ELEMENT test_cookies_time_ok (#PCDATA)>
```

Der eigentliche Exportvorgang beginnt mit dem Abfragen aller Daten aus der Tabelle mit Hilfe eines SQL-Befehls. Dabei wird auch gleich eine Sortierung zur jeweiligen Hierarchie vorgenommen. Für die “user”-Option ist dies folgender Befehl:

```
sqlite3 "$tempdir/checkdb.sqlite" "select ↩
   'finaldb_url'.'systemname', 'finaldb_url'.'username', ↩
   'finaldb_url'.'browsername', 'finaldb_url'.'t1key'
from
'finaldb_url'
order by
'finaldb_url'.'username', 'finaldb_url'.'systemname', ↩
   'finaldb_url'.'browsername', 'finaldb_url'.'ID'
" > "$tempdir/$tempfile"
```

Es werden nur die eindeutige Nummer, der Systemtyp, der Browsername und der Benutzername abgefragt. Diese Ausgaben werden zeilenweise abgearbeitet. Immer dann, wenn ein neuer Name in einem der drei Datenfelder steht, wird die Hierarchie entsprechend angepasst. Die Einträge für die einzelnen Datensätze werden durch weitere SQL-Abfragen anhand des Feldnamens und der eindeutigen Nummer realisiert. Dieses Vorgehen ist bei allen Exportarten - mit Ausnahme der Sortierung des SQL-Befehls und der jeweiligen Struktur - dieselbe.

Wurde beim Programmstart keine Option übergeben, wird automatisch “all” genutzt. Alle erstellten Dateien werden abschließend noch in den Ordner des Programmstarts kopiert und diese Position wird auch auf der Konsole ausgegeben.

Zu beachten ist außerdem, dass die einzelnen URLs von nicht alphanumerischen Zeichen befreit werden müssen, da diese teilweise Probleme in der XML-Datei verursachen können [62]. Es müssen daher folgende Zeichen durch einen so genannten Escape-Code ersetzt werden:

- Kaufmännisches Und-Zeichen (&) durch “&”
- Einfaches Anführungszeichen (’) durch “'”
- Doppeltes Anführungszeichen (“) durch “"”

- Größer-als-Zeichen (>) durch ">"
- Kleiner-als-Zeichen (<) durch "<"

Diese jeweilige Ersetzung wird bei jedem Eintrag nach der Abfrage aus der Datenbank durchgeführt.

6.7.39 Skript: show_all_md5_err

Dieses Skript gibt kurz vor Programmende alle Dateinamen an, bei denen MD5-Fehler während der Bearbeitung aufgetreten sind. Sollten, wie es normalerweise der Fall ist, keine Fehler aufgetreten sein, wird keine Ausgabe dieses Skripts erstellt.

6.7.40 Skript: temp-save

Mit Hilfe dieses Skriptes wird der gesamte temporäre Ordner in ein ZIP-Archiv komprimiert, das wiederum in den Ordner des Programmaufrufs kopiert wird. Dies muss allerdings direkt beim Programmaufruf mit Hilfe des Punktes "SAVETEMP" angegeben werden. Der Ablauf ist dabei folgender:

- Zunächst wird das aktuelle Datum inklusive Uhrzeit als String in eine lokale Variable gespeichert. Dazu wird das Programm "date" genutzt.
- Anschließend wird der Kompressionsvorgang in eine Datei, die aus dem Namen "whft_backup_" mit angehängtem, zuvor erstellten Datumsstring besteht, gestartet.
- Das erstellte ZIP-Archiv wird abschließend mittels "mv" in den Ordner des Programmaufrufs verschoben. Zusätzlich wir die Position auf der Konsole ausgegeben.

Durch die Archivierung wird die Aufzeichnung des Logvorgangs mit der Ausführung des eigentlichen Befehls beendet. Es fehlen daher im Log die Zeilen zum Löschen des temporären Ordners und das Programmende, worauf allerdings verzichtet werden kann.

Die Größe des erstellten Archivs ist dabei abhängig von der Anzahl der gefundenen URLs, dem zugehörigen Cache und der Größe des jeweiligen Downloads. Es sollte daher immer mit mehreren hundert Megabytes gerechnet werden.

6.7.41 Skript: temp-delete

Im letzten Schritt wird vor Beendigung des Programms der temporäre Ordner mit Hilfe dieses Skripts gelöscht. Wurde vom Benutzer keine Archivierung des Ordners mittels “SAVETEMP” gefordert, gibt es außer den XML-Dateien keine verbliebenen Dateien auf der Festplatte.

6.7.42 Skript: show-help

Dieses Skript wird nur dann aufgerufen, wenn der Benutzer das Programm mit der Option “-h” oder “–help” startet. Es wird anschließend der Inhalt der Hilfsdatei ausgegeben. Dabei werden die eingestellte Sprache und die definierte Ausgabeoption (“dialog”, “Xdialog” oder “gdialog”) verwendet. Nach dem Schließen des Fensters wird das gesamte Programm beendet.

7 Ergebnisse

Nachdem im vorherigen Kapitel der Programmaufbau und die Implementierung ausführlich beschrieben wurden, widmet sich dieses Kapitel den Ergebnissen. Im nächsten Unterkapitel werden die Möglichkeiten für die erstellte Anwendung besprochen. Anschließend wird in Kapitel 7.2 auf Seite 107 auf die Bedienung eingegangen. Das darauffolgende Unterkapitel 7.3 auf Seite 113 befasst sich mit der Installation des Programms. Schließlich wird die ebenfalls erstelle Live-CD in Kapitel 7.4 auf Seite 114 beschrieben. Danach werden die durchgeführten Tests in Kapitel 7.5 auf Seite 116 mit den eigens erstellen System-Images besprochen und auf die Stärken (7.6 auf Seite 119) und Schwächen (7.7 auf Seite 120) der Implementierung eingegangen. Das letzte Unterkapitel 7.8 auf Seite 122 befasst sich abschließend mit dem Bezug zur Zielsetzung.

7.1 Eigenschaften

Einige Eigenschaften des Programms wurden bereits in den vorherigen Kapiteln kurz erwähnt. In der nachfolgenden Auflistung werden diese und weitere Punkte besprochen:

- **Reproduzierbarkeit:**
  Einer der wichtigsten Punkte ist die reproduzierbare Ausführbarkeit der Anwendung. Wird das Programm ohne Tests ausgeführt, so sind die Hashwerte der Ergebnisse für ein Image identisch. Dies ist vor allem für die Nutzbarkeit als forensisches Tool, wie in Kapitel 4.1 auf Seite 17 aufgezeigt, notwendig. Sobald allerdings die Tests durchgeführt und die Daten aus dem Internet heruntergeladen werden, ist die Binärkompatibilität der Ausgaben aufgrund von Webseiten mit dynamischen Inhalten, wie etwa einer Uhrzeit, nicht mehr gegeben. Die Einträge selbst sind allerdings weiterhin identisch, die Tests können jedoch bei neuen Onlinedaten andere Ergebnisse liefern.

- **Einfache Konfiguration:**
  Die default-Einstellungen des Programms können einfach eingestellt werden. Dazu muss nur die Datei “config.txt” im Programmunterordner “etc” bearbeitet werden. Der Inhalt dieser Datei ist in Kapitel 6.3.1 auf Seite 43 dargestellt. Die jeweils erlaubten Werte sind für jede Zeile in dem darüberstehenden Kommentar in Klammer aufgeführt. Die unter “default values” stehenden Werte werden genutzt, wenn beim Programmaufruf der entsprechende Parameter nicht übergeben wird.

- **Mehrere Aufrufmöglichkeiten:**
  Der Start des Programms kann über drei Arten erfolgen. Einmal können alle Optionen mit der zu bearbeitenden Quelle direkt übergeben werden. Alternativ kann auch eine Konfigurationsdatei mit den gewünschten Punkten übergeben werden. Die letzte Möglichkeit ist außerdem der Programmaufruf ohne einen einzigen Parameter, wodurch ein interaktives Menü zum Einstellen geöffnet wird. Dieses Menü kann, abhängig von der Konfiguration, auch in einem Fenster grafisch dargestellt werden. Nähere Informationen zu den Aufrufmöglichkeiten sind im nächsten Kapitel 7.2 auf Seite 107 zu finden.

- **Optionaler interaktiver Ablauf:**
  Neben dem interaktiven Start, bei dem die Einstellungen abgefragt werden, kann auch die Programmausführung schrittweise erfolgen. Wird diese Option gesetzt, wartet das Programm nach jedem ausgeführten Punkt auf eine Eingabe des Benutzers. Diese schrittweise Ausführung hat den Sinn, dass bei den einzelnen Schritten die Ausgabe in Ruhe mitgelesen werden kann und bei einer eventuellen Abwesenheit nichts verpasst wird.

- **Optionale grafische Oberfläche:**
  Optional kann für die Konfiguration auch eine grafische Oberfläche genutzt werden. Dazu muss in der Konfigurationsdatei der Punkt “DIALOG” auf “auto”, “Xdialog” oder “gdialog” gestellt werden. Ist das entsprechende Programm nicht installiert, wird automatisch auf “dialog” zurückgegriffen. Abbildung 12 auf Seite 65 zeigt diese unterschiedlichen Ausgabearten an.

- **Vielzahl an Quellen möglich:**
  Als Datenquelle für die Auswertung können sowohl einzelne Partitionen als auch ganze Festplatten genutzt werden. Dabei ist es egal, ob es sich um eine unkomprimierte Image-Datei oder um eine richtige Festplatte handelt. So etwa kann von einer Live-CD aus die Festplatte des jeweiligen Rechners untersucht werden, ohne Daten zu verändern. Außerdem ist es möglich, Festplatten von virtuellen Systemen zu analysieren, sofern diese nicht komprimiert sind. Das Auslesen einer komprimierten oder verschlüsselten Quelle wird nicht unterstützt.

- **Einfache Installierbarkeit:**
  Die Installation der Anwendung gestaltet sich einfach. Das Programm muss nur in

einen beliebigen Ordner kopiert oder entpackt werden. Anschließend müssen die gewünschten temporären Pfade in die Konfiguration eingetragen werden. Genaueres dazu ist im Kapitel 7.3 auf Seite 113 zu finden.

- **Portabel auf Unix-Systemen:**
  Da es sich beim Programm im Wesentlichen um eine Sammlung von Bash-Skripten handelt, ist die Applikation portabel und unabhängig vom jeweiligen Ausführungsort. Damit dies auch funktioniert, wird beim Programmstart die Position des Programmordners gesucht, was abhängig von der Festplattengröße etwas Zeit beanspruchen kann. Für eine fixe Installation empfiehlt es sich, den genauen Pfad in die Datei "start" und das dort enthaltene Feld "SEARCH" einzutragen.

- **Auf einer Live-CD nutzbar:**
  Bei der dieser Arbeit beiliegenden DVD handelt es sich um eine Live-CD, auf welcher sich ebenfalls das Programm befindet. Diese CD kann zum Untersuchen von Rechnern genutzt werden, ohne vorher die Festplatte ausbauen zu müssen. Außerdem muss vom installierten System, sofern die Festplatte nicht verschlüsselt ist, nichts bekannt sein. Nach dem Herunterfahren der Live-CD bleiben auf dem System keine Spuren zurück.

- **Einige Abhängigkeiten:**
  Das Programm besitzt eine Vielzahl von Abhängigkeiten, wobei die meisten davon Standard-GNU-Tools sind und daher auf den meisten aktuellen Linux-Systemen bereits installiert sind. Auf BSD-Systemen lassen sich diese ebenfalls installieren, weshalb die Anwendung dort auch funktioniert. Theoretisch ist auch eine Ausführung unter Windows mit Hilfe von cygwin [12] möglich. Doch dies wurde nie getestet, da unter Windows ein nur lesender Zugriff auf ein Image nicht zuverlässig garantiert werden kann. Weitere Information dazu sind in Kapitel 6.5 auf Seite 52 zu finden.

- **Erweiterbarkeit:**
  Die Anwendung kann relativ einfach erweitert werden, da jeder Schritt zwar auf den vorherigen aufbaut, aber nicht an diesen gebunden ist. Es lassen sich also komplett neue Schritte jederzeit hinzufügen. Sollen hingegen neue Browser unterstützt werden, so müssen mehrere einzelne Skripte angepasst werden. Dasselbe gilt auch für das Hinzufügen weiterer Tests im Bereich der "generate-check-db"-Auswertung.

Einige der angesprochenen Punkte, wie zum Beispiel die Optionen beim Programmstart, werden im Laufe der nächsten Kapitel genauer behandelt.

7.2 Bedienung

Die Bedienung des Programms ist einfach. So gibt es, wie bereits in den vorherigen Kapiteln mehrfach erwähnt, drei Startmöglichkeiten und einige einstellbare Optionen. Die Datei zum Starten der Ausführung nennt sich “start” und ist im Hauptverzeichnis des Programmordners, wie in Kapitel 6.3.1 auf Seite 43 dargestellt, zu finden. Der Aufruf erfolgt dabei, wie jedes Shell-Skript, entweder mit “bash start” oder bei gesetzten “+x“-Flag mit “./start”. Das notwendige Flag kann mit “chmod +x start” gesetzt werden.

Abbildung 13: Startdialog für die interaktive Programmausführung

Es gibt drei Möglichkeiten, wie Optionen beim Programmstart übergeben werden können:

1. **Aufruf ohne Parameter:**
   Zum Beispiel:

```
bash start
```

   In diesem Fall wird der interaktive Modus gestartet und es erscheint das in Abbildung 13 auf der vorherigen Seite dargestellte interaktive Menü. Über dieses können nun die einzelnen Optionen definiert oder über eine Konfigurationsdatei geladen werden. Die jeweilige Konfiguration kann über den dritten Menüpunkt jederzeit betrachtet werden. Der vierte und fünfte Punkt startet die Verarbeitung normal oder interaktiv. Zusätzlich kann vom Menü aus auch die Hilfe aufgerufen werden.

2. **Aufruf über eine Konfigurationsdatei:**
   Zum Beispiel:

```
bash start -c config.txt
```

   Hier wird die Datei "config.txt" zum Übergeben der Parameter genutzt. Der Aufbau einer solchen Datei besteht nur aus den möglichen Parametern mit der jeweiligen Konfiguration. Eine mögliche Konfiguration ist zum Beispiel:

```
1  ### config file for whft
2  SOURCE=/home/user/images/MSC-Debian.img
3  USER=all
4  BROWSER=firefox,opera,ie,chrome
5  SEARCHFILES=no
6  CHECK=yes
7  COOKIES=yes
8  OUTPUT=all
9  SAVETEMP=yes
10 INTERACTIVE=no
```

   Es wird also das Image "MSC-Debian.img" genutzt und die Variablen entsprechend der Optionen eingestellt.

3. **Aufruf über direkte Optionen:**
   Zum Beispiel:

```
# lange Schreibweise
bash start --source=/home/user/images/MSC-Debian.img ↩
   --user=all --browser=chrome,opera --searchfiles=yes ↩
   --check=yes --cookies=yes --output=one --savetemp=yes ↩
   --interactive=no
# kurze Schreibweise
bash start -s /home/user/images/MSC-Debian.img -u all -b ↩
   chrome,opera -sf yes -ch yes -co yes -o one -st yes -i no
```

   Die beiden Befehle übergeben dieselben Optionen. Diese "kurze" und "lange" Schreibweise kann auch kombiniert werden. Wird allerdings "-h", "–help", "-c" oder "–configfile" mit angegeben, so wird davon die erste angegebene Option genutzt und alle anderen Parameter werden ignoriert.

Diese drei Aufrufmöglichkeiten können alle dieselben Optionen nutzen. In der nachfolgenden Auflistung sind alle Möglichkeiten zu finden:

- **Zu nutzende Datenquelle:**
  Mit dieser Option kann eingestellt werden, welche Datei oder welches Device bearbeitet werden soll. Bei Problemen mit dem relativen Pfad muss der vollständige Pfad vom Wurzelverzeichnis aus angegeben werden. Die Parameter werden folgendermaßen übergeben:

```
# Aufruf mit Parameterangabe:
-s <Pfad> oder --source=<Pfad>
# Zeile in der Konfigurationsdatei:
SOURCE=<Pfad>
```

- **Zu bearbeitende Benutzer:**
  Hier können die gewünschten Benutzernamen angegeben werden. Die Trennung erfolgt dabei durch einen Beistrich. Wird hier nur "all" eingetragen, so werden alle gefundenen Benutzer bearbeitet. Die Aufrufmöglichkeiten sind:

```
# Aufruf mit Parameterangabe:
-u <Namen> oder --user=<Namen>
# Zeile in der Konfigurationsdatei:
USER=<Namen>
```

- **Zu bearbeitende Browser:**
  Hier werden die zu nutzenden Browser eingetragen. Als Trennzeichen dient ebenfalls ein Beistrich. Das Wort "all" bewirkt, dass alle möglichen Browser verarbeitet werden. Mögliche Namen sind "firefox", "opera", "chrome" und "ie". Übergeben wird dies folgendermaßen:

```
# Aufruf mit Parameterangabe:
-b <Namen> oder --browser=<Namen>
# Zeile in der Konfigurationsdatei:
BROWSER=<Namen>
```

- **Suche von gelöschten Dateien:**
  Mit dieser Option wird die Datenquelle auf gelöschte Dateien hin untersucht. Akzeptiert wird dabei nur der Wert "yes" oder "no". Die Angabe erfolgt folgendermaßen:

```
# Aufruf mit Parameterangabe:
-sf <yes/no> oder --searchfiles=<yes/no>
# Zeile in der Konfigurationsdatei:
SEARCHFILES=<yes/no>
```

- **Durchführung von Tests:**
  Hier kann eingestellt werden, ob die gefundenen URLs getestet werden sollen. Wird hier "no" gesetzt, so sind in der XML-Datei die zugehörigen Felder auch nicht aufgeführt. Es kann nur "yes" oder "no" angegeben werden. Wird "no" gesetzt, werden auch keine Cookies verarbeitet. Die Übergabe erfolgt folgendermaßen:

```
# Aufruf mit Parameterangabe:
-ch <yes/no> oder --check=<yes/no>
# Zeile in der Konfigurationsdatei:
CHECK=<yes/no>
```

- **Cookies Verarbeitung:**
  Die Bearbeitung von Cookies kann mit dieser Option explizit abgeschaltet werden. Dadurch werden nur mehr vier der sieben Tests durchgeführt. Angegeben kann ebenfalls nur "yes" oder "no" werden. Eingestellt wird diese Option durch:

```
# Aufruf mit Parameterangabe:
-co <yes/no> oder --cookies=<yes/no>
# Zeile in der Konfigurationsdatei:
COOKIES=<yes/no>
```

- **Zu erstellende Ausgabearten:**
  Über diese Option kann die Ausgabe des Programms eingestellt werden. Möglich ist dabei "all", "user", "browser", "single" und "one". Die Bedeutung der einzelnen Einträge kann in Kapitel 6.7.38 auf Seite 97 nachgelesen werden. Übergeben werden diese Informationen folgendermaßen:

```
# Aufruf mit Parameterangabe:
-o <Art> oder --output=<Art>
# Zeile in der Konfigurationsdatei:
OUTPUT=<Art>
```

- **Archivierung des Arbeitsordners:**
  Damit wird eine Sicherung des temporären Arbeitsordners erstellt. Als Optionen sind nur "yes" oder "no" möglich. Bei "yes" wir der Ordner komprimiert als ZIP-Datei im Ausführungsordner hinterlegt. Übergeben wird diese Option folgendermaßen:

```
# Aufruf mit Parameterangabe:
-st <yes/no> oder --savetemp=<yes/no>
# Zeile in der Konfigurationsdatei:
SAVETEMP=<yes/no>
```

- **Schrittweise Programmausführung:**
  Über diese Option kann die schrittweise Ausführung eingestellt werden. Möglich sind die Werte "yes" und "no". Wenn "no" eingestellt ist, läuft das Programm bis zum Ende ohne weitere Eingaben durch. Eingestellt kann dies folgendermaßen werden:

```
# Aufruf mit Parameterangabe:
-i <yes/no> oder --interactive=<yes/no>
# Zeile in der Konfigurationsdatei:
INTERACTIVE=<yes/no>
```

Sämtliche soeben besprochene Möglichkeiten sind auch in der Hilfedatei hinterlegt. Diese kann über "-h" oder "–help" betrachtet werden. Dafür existiert ebenfalls ein Eintrag im Menü.

Neben den besprochenen Punkten kann das Programm außerdem noch über die Konfigurationsdatei eingestellt werden. Die Datei befindet sich im Unterordner "etc" und nennt sich "config.txt". Der Inhalt ist in Kapitel 6.3.1 auf Seite 43 dargestellt. Dort können neben den Standard-Werten für die eben genannten Optionen auch die Pfade, die Sprache, die Ausgabeart und weitere Optionen eingestellt werden. Die jeweils erlaubten Optionen sind in dem dazugehörigen, darüberstehenden Kommentar angegeben. Bei den Pfaden muss der Weg mit Ausnahme des letzten Ordners existieren. Bei den Dateinamen sind alle Buchstaben und Zahlen bis auf einige Sonderzeichen und Leerzeichen erlaubt.

Aufgrund der definierten Standard-Werte ist es auch möglich, dass Parameter einfach ausgelassen werden können, ohne dabei den Programmablauf zu behindern. Der minimalste Programmaufruf ohne Startmenü ist daher folgender:

```
bash start -s /home/user/images/MSC-Debian.img
```

Bei häufiger Verwendung gewisser Optionen empfiehlt sich daher die Anpassung der Konfigurationsdatei.

7.3 Installation

Die Installation des Programms gestaltet sich relativ einfach. Man muss einfach den Ordner in ein beliebiges Verzeichnis kopieren oder entpacken, und schon kann das Programm, sofern die Abhängigkeiten erfüllt sind, ausgeführt werden. Es müssen daher die notwendigen Programme installiert und die Berechtigungen für das Mounten erteilt werden. Wie dies funktioniert, wird in Kapitel 6.5 auf Seite 52 beschrieben. Zusätzlich sollte die Konfigurationsdatei im Unterordner "etc" einmalig bearbeitet und angepasst werden.

Da das Programm, in der auf der DVD beigelegten Version, das gesamte lokale Dateisystem nach den Programmordner durchsucht, empfiehlt es sich, den Ordner einmalig einzustellen. Dazu muss die Datei "start" bearbeitet und der Start des Suchvorganges bei der Option "SEARCH" am Anfang der Datei eingestellt werden. Je exakter die Pfadangabe ist, desto schneller erfolgt der Start. Bei der auf der Live-CD installierten Version wurde "/opt/whft" eingetragen, wodurch der Start in etwa einer Sekunde erfolgt. Der Suchvorgang ist notwendig, um den exakten Ordner festzustellen und damit Programmfehler aufgrund eines falschen Pfades zu vermeiden. Wurde der Pfad einmal gefunden, erfolgt jeder weitere Programmstart bis zum Schließen der Konsole ohne Zeitverlust.

Bei einer fixen Installation auf einem Linux-System empfiehlt sich folgendes Vorgehen:

1. Kopieren des Programmordners nach "*/opt/whft*".
2. Anpassen der "start" an den genauen Standort.
3. Erteilen der Ausführungsberechtigung mittels "*chmod +x start*".
4. Anpassen der "config.txt" an die gewünschten Eigenschaften.
5. Erstellen eines symbolischen Links von "*/opt/whft/start*" in ein Verzeichnis, welches für die Ausführung von Programmen zuständig ist. Dies kann "*/usr/local/bin*"

oder "*/usr/bin*" sein. Zusätzlich empfiehlt es sich, "start" als "whft" zu verlinken. Ein möglicher Befehl für diesen Zweck ist:

```
ln -s /opt/whft/start /usr/local/bin/whft
```

Sind diese Schritte abgeschlossen, kann das Programm von jedem Ordner aus mittels "whft" gestartet werden. Erstellte XML-Dateien und Sicherungen des Programmordners werden dabei in den Pfad des jeweiligen Aufrufes kopiert. Auf der beiliegenden Live-CD wurde das Deaktivieren der Portabilität und damit verbundenen fixen Installation genau auf diese Weise durchgeführt.

Auf der dieser Arbeit beiliegenden DVD ist ebenfalls ein kleines Installationsskript zu finden. Dieses muss mit der Angabe des gewünschten Installationspfades ausgeführt werden.

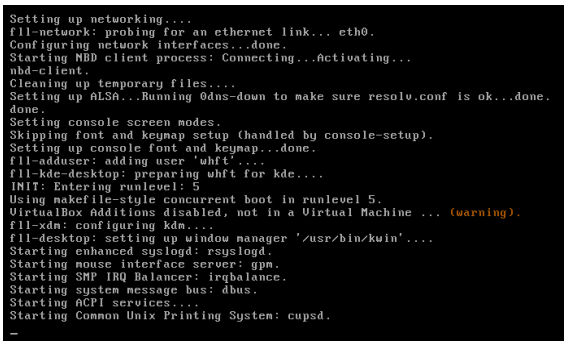
7.4 Live-CD

Die dieser Arbeit beigelegte DVD kann auch als Live-CD genutzt werden. Gestartet wird das Programm, indem die DVD in das Laufwerk des jeweiligen PCs eingelegt und davon gebootet wird. Dazu muss entweder im BIOS die Bootreihenfolge definiert oder der bei neueren Rechnern vorhandene Abfragedialog aufgerufen werden. Häufige dafür verwendete Tasten sind "ESC", "F9" oder "F12". Dies Tastenbelegung ist abhängig vom jeweiligen Hersteller.

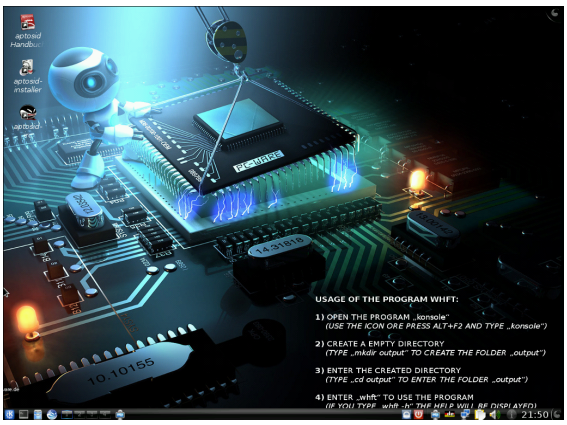
Sobald der Start von der DVD erfolgt, erscheint der Bootmanager, wie in Abbildung 14(a) auf der nächsten Seite dargestellt. Anpassungen der Startparameter müssen im Normalfall nicht vorgenommen werden. Sollte entgegen den Erwartungen der Monitor beim Laden der grafischen Oberfläche schwarz bleiben, muss "vesa" als Treiber genutzt werden. Dies wird mittels "*xmodule=vesa*" [63] in der Befehlszeile des Bootmanager GRUB angehängt. Weitere Bootoptionen sind in dem der Live-CD beiliegenden Handbuch zu finden.

Nach dem Bootmanager startet das System in die grafische Oberfläche. Abbildung 14(b) auf der nächsten Seite zeigt einen Ausschnitt aus dem Bootvorgang an. Sobald das Bild wie in Abbildung 14(c) aussieht, ist der Startvorgang abgeschlossen.

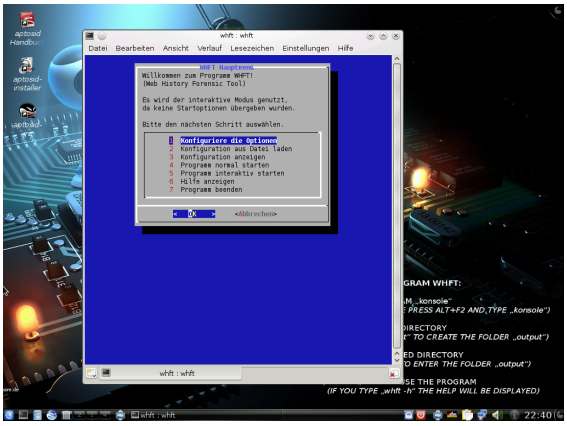
(a) Bootmanager



(b) Startvorgang



(c) Grafische Oberfläche



(d) Gestartetes Programm

Abbildung 14: Screenshots der Live-CD

Das im Zuge der Arbeit erstellte Programm kann nun in jeder Konsole mit dem Befehl “whft” aufgerufen werden. Zusätzlich befindet sich ein Icon auf der Arbeitsfläche und im Startmenü, welches das interaktive Menü öffnet. In Abbildung 14(d) ist das über das Icon gestartete Programm zu sehen.

Für die Bedienung der Live-CD ist ebenfalls ein Handbuch des aptosid-Teams [64] enthalten. Zum Öffnen des Handbuchs existiert ebenfalls ein Icon auf dem Desktop. Darin sind unter anderem Informationen zur Netzwerkkonfiguration und Installation zu finden.

7.5 Tests

Das Durchführen von Tests ist ein wichtiger Punkt für ein stabiles Produkt. Deshalb wurde bereits bei der Implementierung jedes einzelne Skript mit mehreren unterschiedlichen Quellen getestet. Insgesamt kamen dabei folgende fünf Datenquellen zu Einsatz:

1. Es wurden ein Partitionsimage und ein Diskimage aus einer alten 10-Gigabyte-Festplatte mit Windows XP erstellt. Auf diesem System war nur der alte Internet-Explorer 6.0 und Opera 9.20 installiert. Außerdem war lediglich das ServicePack 2 für XP vorhanden.

2. Es wurde ein virtuelles System mit Windows XP und ServicePack 3 erstellt. Außerdem wurden mit jedem unterstützten Browser einige Webseiten besucht. Bei den Browsern handelte es sich um den InternetExplorer 8, Opera 10.62, Firefox 3.6 und Chrome 6.0, da diese zum Zeitpunkt der Installation gerade verfügbar waren.

3. Zudem wurde ein weiteres virtuelles System mit Windows 7 erstellt. Auch hier wurden mehrere Webseiten pro Browser besucht. Da das System am selben Tag wie das virtuelle XP installiert wurde, handelte es sich um dieselben Browserversionen.

4. Zusätzlich erstellt wurde ebenfalls ein virtuelles Debian-Squeeze. Hier wurden alle Browser, mit Ausnahme des InternetExplorers, zum Surfen genutzt. Bei den genutzten Browsern handelte es sich um den Firefox 3.5.11, Opera 10.60 und Chrome 6.0, da diese zu dem Zeitpunkt über “apt-get” aus den jeweiligen Repositories installierbar waren.

5. Genutzt wurde auch die Festplatte des Rechners, auf dem das Programm erstellt wurde. Die dort installierten Browserversionen änderten sich kontinuierlich während der Entwicklung. Die aktuellsten Tests wurden mit Firefox 4.0 Beta, Opera 10.64 Beta, Chrome 8.0 Beta und InternetExplorer 9 durchgeführt. Die Auswertung erfolgte dabei ohne Probleme, da die interne Struktur nicht geändert wurde.

Erst wenn ein Skript alle fünf Testarten erfolgreich bearbeitet, gilt es als fertiggestellt. Dass dieses Vorgehen wichtig war, zeigte sich beim Image des nicht virtualisierten Windows XP. Dort stürzte das Programm “pasco” reproduzierbar bei einer “index.dat” ab. Die zugehörige Fehlermeldung ist in Kapitel 6.7.22 auf Seite 83 dargestellt. Dies war

auch der Grund, weshalb das Skript “generate-work-db-ie-pasco” erstellt wurde. Dieser Fehler ließ sich allerdings bei keinen weiteren Test mit einer anderen “index.dat” reproduzieren.

Der Vergleich zwischen der mit dem Programm erstellten Ausgabe und den Daten in der jeweiligen History-Datei wurde händisch durchgeführt. Für das Betrachten der SQLite-Datenbanken von Firefox oder Chrome wurden die Programme “sqliteman” [65] und “sqlitebrowser” [66] verwendet. Für die Dateien des Opera wurde ein einfacher Texteditor genutzt. Beim InternetExplorer kamen die in Kapitel 3.5 auf Seite 9 vorgestellten Tools zum Einsatz. Programme für Windows wurden dabei unter Linux mit Hilfe von “wine” [67] ausgeführt. Verglichen wurden dabei die Anzahl der Einträge, die jeweiligen URLs, die Titel und die Zeitpunkte. Existierten Unterschiede, wurden die betreffenden Fehler lokalisiert und anschließend behoben.

Nach der Implementierung wurden mehrere Testdurchläufe mit dem Programm durchgeführt, wobei noch einige kleine Fehler entdeckt wurden. Diejenigen URLs, die in den einzelnen History-Dateien stehen, wurden dabei immer mit denen der fertig erstellten Datenbank verglichen. Diese Tests basierten auf Kombinationen von mehreren Systemen und dem Verschieben von Browserordnern außerhalb der normalen Position. Außerdem wurde das Programm immer wieder im interaktiven Modus angehalten, um Dateien zu manipulieren. Im Anschluss daran wurde das weitere Verhalten beobachtet. Im Folgenden werden einige dieser durchgeführten Tests beschrieben:

- **MD5-Fehlererkennung:**
  Um die Logik zur Erkennung der MD5-Fehler zu testen, wurde die Anwendung im interaktiven Modus gestartet. Direkt nach dem Kopiervorgang wurden Dateien verändert und Einträge manipuliert. Als das Programm anschließend fortgesetzt wurde, erkannte es zuverlässig alle betroffenen Dateien und verschob diese in den Order “err”. Allerdings werden Manipulationen, welche direkt während des Auslesevorganges stattfinden, nicht erkannt. Für die Praxis dürfte dies hingegen irrelevant sein, da der Rechner, auf dem die Auswertung läuft, normalerweise vor unerlaubten Zugriff geschützt ist und der Auslesevorgang weniger als eine Minute benötigt.

- **Reproduzierbarkeit:**
  Dasselbe Image wurde immer wieder getestet und auf Unterschiede bei den Ergebnissen untersucht. Es hat sich dabei herausgestellt, dass die Reproduzierbar-

keit für die Anzahl der Einträge und die auslesbaren Felder gegeben ist. Anders sieht es hingegen bei den Tests aus, bei denen mit der heruntergeladenen Datei gearbeitet wird. Diese kann bei jedem Download anders sein und dadurch das Testergebnis verändern. Wird die Anwendung direkt hintereinander ausgeführt, sind im Normalfall keine Unterschiede bei den Tests zu erwarten. Liegt zwischen der Ausführung hingegen ein Zeitraum von mehreren Wochen, treten die erwähnten Probleme teilweise auf.

- **Multibootsysteme:**
  In einem weiteren Test wurde ein Image mit mehreren Systemen bearbeitet. Dazu wurde das Skript im interaktiven Modus gestartet und nach der Ausführung des Skripts "mount-image" angehalten, um weitere Images zu mounten. Alle im ersten Teil dieses Kapitels erwähnten Images wurden anschließend eingebunden und dadurch eine Multibootumgebung simuliert. Für die restliche Ausführung der Anwendung spielt die Anzahl der eingebundenen Partitionen nämlich keine Rolle. Nach der Beendigung wurden die Ergebnisse mit denen der einzelnen Ausführungen verglichen. Es stellte sich heraus, dass sämtliche Einträge zuverlässig erkannt und bearbeitet wurden. Ebenso funktionierte die Zuordnung des Systemtyps ohne Probleme.

- **Mehrere Browserordner pro User:**
  Ein weiterer Test bestand darin, dass mehrere Browserordner für einen Benutzer angelegt wurden. Die Erkennung und Zuordnung funktioniert problemlos, solange sich der Ordner im jeweiligen Benutzerordner befindet. Liegt dieser außerhalb, funktioniert zwar die Erkennung, aber nicht die Zuordnung. Der Benutzername für die betreffenden URLs wird dadurch mit "undefined" und einer laufenden Nummer pro History-Ordner in die Datenbank eingetragen. Dasselbe gilt für den Systemtyp, wenn dieser nicht zuordenbar ist. Es war daher zu sehen, dass alle Browserordner ausgewertet und in die Datenbank eingetragen werden.

- **Wiederherstellen von Dateien:**
  Dieser Test wurde, aufgrund der langen Dauer von mehreren Stunden, nur wenige Male durchgeführt. Die bewusst gelöschten Dateien wurden zuverlässig wiederhergestellt und in die Verarbeitung aufgenommen. Als Systemtyp wurde dabei wie

vorgesehen “recovered” und als Benutzername der jeweilige Dateiname eingetragen. In der Bashlog-Datei erscheint nur eine minimale Ausgabe des Programms “photorec”, da dieses den Konsolenframebuffer nutzt und die Ausgabe ständig aktualisiert. Das genaue Log zum Wiederherstellungsvorgang ist in der Datei “photorec.log” im temporären Verzeichnis zu finden.

Die erwähnten Tests und die Einzeltests der jeweiligen Images sind auf der beigelegten DVD als Sicherung des temporären Arbeitsordners hinterlegt. Darin sind neben den XML-Dateien, Datenbanken und Browserverzeichnissen auch die heruntergeladenen Dateien zu finden. Zu beachten ist, dass kein Virenscanner auf die Dateien zugreift, um die Daten nicht zu verändern. Die einzelnen Systeme wurden zwar vor der Virtualisierung einem vollständigen Systemscan unterzogen, dennoch kann eine totale Virenfreiheit nicht vollständig garantiert werden.

Die einzelnen URL-Tests wurden stichprobenartig händisch überprüft. Dabei wurde die jeweilige Webseite mit Firefox 3.6 und einem geleerten Cache besucht. Anschließend wurden die gesetzten Cookies mit denen aus der “workdb.sqlite” verglichen. Dasselbe galt auch für den Titel der Webseite und den jeweiligen Hashwert.

7.6 Stärken

Die Eigenschaften des Programms sind im wesentlichen auch seine Stärken:

- Dem Benutzer wird eine vielfältige Eingabe- und Konfigurationsmöglichkeit geboten. Dabei wird aber auch gleich eine Fehlerkorrektur durchgeführt. Genaueres dazu ist in Kapitel 7.2 auf Seite 107 zu finden.
- Die einzelnen Abläufe des Programms sind aufeinander aufbauend, jedoch nicht direkt abhängig. Dadurch können neue Skripts einfach in den Ablauf miteingebunden werden.
- Durch die Nutzung von Linux als Basissystem können alle aktuellen Dateisysteme genutzt werden. Über den optionalen Suchvorgang ist ebenso eine Nutzung von defekten Partitionen, allerdings ohne die Cookie- und Cacheverarbeitung, möglich.
- Ebenfalls kann ein garantierter Schreibschutz für das Image auf Kernelbasis sichergestellt werden, wodurch forensische Untersuchungen erst möglich werden.

- Durch die Möglichkeit, die Anwendung mit einer Live-CD zu kombinieren, kann auch ein Rechner oder Laptop ohne das vorherige Ausbauen der Festplatte untersucht werden.

- Da das Programm unabhängig vom jeweils im Image installierten System ist, gibt es keine Einschränkungen beim Durchsuchen des Dateisystems. Ebenso werden "versteckte" Dateien und Ordner direkt gefunden und können verarbeitet werden.

- Der Ablauf ist komplett automatisiert und benötigt keine grafische Oberfläche. Daher können mittels Batchverarbeitung mehrere Images hintereinander ohne Benutzerinteraktion abgearbeitet werden. Ein einfaches Skript für eine solche Verarbeitung wäre:

```
1 i=0
2 while read LINE
3 do
4   mkdir i
5   cd i
6   whft -s $LINE
7   cd ..
8   i=$(($i+1))
9 done < liste-image.txt
```

  Damit wird ein Ordner mit einer laufenden Nummer für jede Zeile in der Datei "liste-img.txt" erstellt und darin die Ausgabe des im Zuge der Arbeit erstellten Programms gespeichert. Auf ähnliche Weise könnte auch eine automatisierte Verarbeitung auf einem Server erfolgen, sobald ein neues Image entdeckt wird.

7.7 Schwächen

Neben den im vorherigen Kapitel aufgeführten Stärken existieren auch Schwächen:

- Die im Image vorkommenden symbolischen Links können nicht verarbeitet werden, wenn diese auf eine andere Partition zeigen. Dies beruht darauf, dass die Struktur nach dem Mountvorgang nicht identisch mit dem des jeweiligen Systems ist. Es

werden zwar alle Ordner verarbeitet, aber eine eventuelle Zuordnung zu einem Benutzer kann dadurch nicht mehr erfolgen.

- Eine Unterscheidung zwischen Systemen desselben Typs in einer Multibootumgebung ist nicht möglich. Dies beruht ebenfalls darauf, dass eine Zuordnung der jeweiligen Benutzer zu einem System eines Typs nicht zuverlässig funktioniert.

- Daher ist der Programmaufbau zwar ziemlich robust, mögliche, nicht vorhergesehene Fehler, können jedoch nicht immer zuverlässig erkannt werden. Diese werden zwar auf der Konsole ausgegeben, aber anschließend nicht weiter behandelt. In einem einzigen Fall kam es bei den Tests zum Absturz des Programms “pasco” ein einer “index.dat”. Die URLs bis zu dem Absturz können zwar verarbeitet, doch darauffolgende nicht mehr berücksichtigt werden. Ebenso ist eine Erkennung dessen, um wie viel Zeilen es sich handelt, nicht möglich. Genauere Informationen zu diesem Fehler sind in Kapitel 6.7.22 auf Seite 83 zu finden.

- Bei der Planung wurden auch einige Skripts entworfen, die auch leicht ohne Funktionsverlust eingespart werden könnten. Dazu gehören zum Beispiel “generate-work-db-undefined” oder “generate-work-db-recovered”. Dafür könnten Inhalte anderer Skripts zu einem neuen zusammengefasst werden, wie es bei den Eingabetests der einzelnen Initialisierung-Skripts möglich wäre. Dies beruht darauf, dass während der Implementierung immer wieder auf Lösungen gestoßen wird, an die in der Planungsphase nicht gedacht wurde.

- Eine Optimierung bei der Ausführungsgeschwindigkeit wurde nicht vorgenommen, da ein Großteil der benötigten Zeit dem Durchsuchen des Dateisystems, dem Kopiervorgang des Browserordners und dem Herunterladen der URL zugerechnet werden kann. Die für die anderen Abläufe genutzte Zeit wurde daher vernachlässigt. Um den Programmablauf zu beschleunigen, empfiehlt sich die Ausführung auf einem Rechner mit einer schnellen Festplatte und guter Internetanbindung.

Die erwähnten Schwächen beeinflussen entweder nicht die Funktionalität des Programms oder können designbedingt nicht geändert werden. Ein Schutz vor einem möglichen Verlust der History-Daten wurde so gut wie möglich implementiert. Fehler benötigter externer Programme konnten dabei leider nicht behoben werden.

7.8 Bezug zur Zielsetzung

Die Ziele, die in Kapitel 5 auf Seite 32 angeführt sind, konnten vollständig erfüllt werden. Im Folgenden werden die vorgegebenen fixen Anforderungen kurz besprochen:

- **URL:**
  Es werden alle URLs der vier unterstützten Browser gefunden und - sofern möglich - dem jeweiligen Benutzer zugeordnet.

- **History:**
  Als Datenquelle dienen, wie gefordert, die History-Dateien der jeweiligen Browser. Bis auf den InternetExplorer werden diese auch ohne zusätzliche Programme ausgelesen.

- **Cache:**
  Die einzelnen Einträge werden im Cache gesucht und die Ergebnisse für den Test herangezogen. Eine genaue Analyse der einzelnen Dateien findet allerdings nicht statt.

- **Cookies:**
  Die gefundenen Cookies werden bei den Tests, soweit es möglich ist, den einzelnen Websites zugeordnet. Die entsprechende Anzahl wird während der Verarbeitung ausgegeben.

- **Zeitlinie:**
  Durch die Nutzung einer SQLite-Datenbank erfolgt die zeitliche Sortierung direkt beim Auslesen der Daten mittels SQL-Befehl.

- **Sessions:**
  Anhand der Zeitlinie können die Einträge zu einzelnen Sitzungen zusammengefasst werden. Der Zeitpunkt, ab wann ein Eintrag als neue Sitzung gilt, kann in der Konfiguration eingestellt werden.

- **Überprüfung:**
  Die jeweils ausgelesene URL wird heruntergeladen und mit den vorhandenen Daten verglichen. Die Ergebnisse der Tests werden in die XML-Datei mit exportiert.

- **Ausgabe:**
  Wie gefordert wurden alle gewonnenen Daten in eine XML-Datei eingefügt. Dabei kann zwischen einer oder mehreren Arten gewählt werden. Zusätzlich existiert aber noch die SQLite-Datenbank mit den identischen Einträgen in der Sicherung des Arbeitsordners.

- **Textbasiert:**
  Die Ausführung ist komplett ohne grafische Oberfläche möglich. Für die Konfiguration kann aber eine optionale grafische Ausgabe genutzt werden.

Bei den zusätzlich gesetzten Zielen verhält es sich folgendermaßen:

- **Mehrere Startmöglichkeiten:**
  Diese wurden erfolgreich integriert. Es ist neben der direkten Angabe der Parameter auch eine Konfigurationsdatei oder interaktive Abfrage möglich.

- **Konfigurierbarkeit:**
  Es wurden mehrere Möglichkeiten zur Konfiguration eingebaut, mit denen neben den Orten für die Daten und den Standard-Werten auch einzelne Parameter definiert werden können.

- **Auswahlmöglichkeiten:**
  Mittels einstellbarer Optionen kann bei jedem Programmaufruf das Programmverhalten beeinflusst werden. Alternativ werden die in der Konfiguration definierten Standard-Werte genutzt.

- **Gelöschte Dateien:**
  Dieses Ziel konnte durch das externe Programm “photorec” erreicht werden. Die auf diese Weise wiederhergestellten Dateien werden entsprechend markiert verarbeitet.

- **XML-Ausgabe:**
  Wie bereits bei den fixen Vorgaben erwähnt, gibt es mehrere Möglichkeiten, die XML-Dateien zu exportieren. Diese beziehen sich auf den Aufbau der Hierarchie innerhalb der Datei. Die Ausgabearten können auch kombiniert werden.

- **Festplatten:**
  Durch die Nutzung von Linux als Basissystem kann dieses Ziel ebenfalls erfüllt werden. Möglich sind ebenso unkomprimierte Festplatten von virtuellen Systemen.

- **Live-CD:**
  Das fertige Programm wurde in eine Live-CD integriert. Diese ist auch auf der dieser Arbeit beigelegten DVD zu finden.

Zusammenfassend kann festgehalten werden, dass alle Vorgaben und Ziele erfüllt wurden. Während der Implementierung stellte sich ebenfalls heraus, dass die zusätzlich gesetzten Ziele den Komfort und die Möglichkeiten auch durch teilweise relativ geringe Anpassungen stark vergrößerten.

Das erstellte Programm verarbeitet zuverlässig die ihm übergebenen Datenquellen und erstellt daraus wie gefordert XML-Dateien. Allerdings gibt es durchaus noch einige Verbesserungs- und Erweiterungsmöglichkeiten, an denen voraussichtlich noch gearbeitet wird. Genaueres dazu ist in Kapitel 8.2 auf Seite 126 zu finden.

8 Diskussion

In den vorherigen Kapiteln 6 auf Seite 35 und 7 auf Seite 104 wurden der Programmaufbau und die Ergebnisse der Arbeit besprochen. In diesem Kapitel wird zunächst die eigene Meinung dargelegt und im Anschlus daran die Erweitungsmöglichkeit behandelt.

8.1 Eigene Meinung

Die gestellte Aufgabe zur automatisierten Auswertung von Browser-History-Daten war für mich sehr interessant und lehrreich. Ich benötigte zwar aufgrund eines ungeplanten Mehraufwandes während des Sommersemesters 2010 und zusätzlichen Problemen während der Implementierung länger als ursprünglich geplant, jedoch hat sich dies ausgezahlt. Die Anwendung funktioniert zuverlässig und arbeitet im Wesentlichen mit allen Arten von unkomprimierten Datenträgern und Abbildern.

Während der Implementierung stieß ich auch auf ein selbstverschuldetes Problem bei der Planung. Dadurch musste ich die komplette Struktur umstellen und verzichtete im Zuge dessen auch auf einzelne Teilbereiche in Java. Zusätzlich wurde die Speicherung der einzelnen Daten in eine SQLite eingeplant. Diese Änderung machte sich bezahlbar, weil nur durch die neu geplante Datenbank eine effiziente Verarbeitung möglich war.

Mit dem Schreiben dieser Arbeit bin ich außerdem auf einige kleine noch existierende Strukturprobleme gestoßen, die unter Umständen eine Einarbeitung weiterer Entwickler erschweren könnten. Als einer der nächsten Schritte ist eine Behebung geplant. Weitere Informationen zur Erweiterungsmöglichkeit und den geplanten folgenden Schritten sind im nachfolgenden Kapitel 8.2 auf der nächsten Seite zu finden.

Das Beschäftigen mit der Browserforensik war außerdem sehr interessant und zeigte, wie unterschiedlich die einzelnen Hersteller ihre Browser implementieren. Mich überraschte vor allem bei einigen Browsern die Menge der gespeicherten Informationen über eine einzelne Website. So speichert etwa der InternetExplorer weit über 200 unterschiedliche Adressen, auch wenn damit nur die drei anderen Browser heruntergeladen werden. Beim anschließenden Durchlesen dieser Adressen sind viele Links zu Seiten vorhanden, die weder bewusst besucht oder als Link auf der Webseite gelesen werden können. Dies zeigt auch, wie gering sowohl die Kontrolle als auch das Wissen über die internen Abläufe

und die Kommunikation des Browsers meist vorhanden sind. Dasselbe gilt auch für die Cookies, denn diese werden während des üblichen Websurfens massenhaft gesetzt, viele davon auch dauerhaft. Plugins zur Verbesserung der Privatsphäre sind daher empfehlenswert.

Forensik im Allgemeinen ist ein sehr interessantes Gebiet. Die Auseinandersetzung mit dieser Thematik führt zu einem größeren Einblick in die jeweilige Materie. Viele Vorgänge und Situationen, die selbstverständlich scheinen, können plötzlich aus einem ganz anderen Blickwinkel gesehen werden. Unabhängig von meinem späteren Beruf werde ich mich daher auch weiterhin mit dem Thema Forensik in Bezug auf IT-Gebiete beschäftigen.

8.2 Erweiterungsmöglichkeiten

Geplant ist, dass das erstellte Programm auch in Zukunft aktiv weiterentwickelt wird. So wird es, wie bereits im vorherigen Kapitel kurz erwähnt, eine kleine Strukturänderung geben. Dadurch soll die Erweiterbarkeit für neue Browser und Startparameter stark vereinfacht werden.

Mit dem Stand des Programms, welches dieser Arbeit beiliegt, können Erweiterungen folgendermaßen realisiert werden:

- **Neue Startoptionen:**
  Um dies zu bewerkstelligen, müssen die neuen Punkte in allen drei Initialisierungs-Skripten implementiert werden. Dasselbe gilt auch für die Tests und Default-Werte in der Konfigurationsdatei.

- **Weiter Browser:**
  Um einen weiteren Browser, wie etwa den Browser Safari, hinzuzufügen, müssen im Wesentlichen alle Skripte ab der Suche der Browserordner angepasst und erweitert werden. Zusätzlich werden bei der Erstellung der Datenbanken neue Skripte benötigt.

- **Zusätzliche URL-Tests:**
  Sollen neue Tests hinzugefügt werden, müssen die “generate-check-db-*” Skripte angepasst werden. Dazu ist es zunächst notwendig, die SQL-Befehle um die

entsprechenden Felder zu erweitern. Anschließend ist das zum jeweiligen Browser gehörende Skript um den eigentlichen Test zu ergänzen. Schließlich muss das neue Feld auch dem XML-Exportskript hinzugefügt werden.

- **Stapelverarbeitung:**
  Dies könnte als eine neue Option implementiert werden. Dazu sind im Wesentlichen nur die drei Startskripte für einen rekursiven Aufruf anhand der übergebenen Liste anzupassen.

- **Neue Funktionen:**
  Neue Skripte können einfach in das Skript "main" eingefügt werden. Denkbar wäre zum Beispiel eine Benachrichtigung über Mail, sobald eine Auswertung beendet ist. Die Position für ein solches Skript würde sich dann direkt vor dem Programmende befinden.

- **Austausch von Skripten:**
  Dies ist prinzipiell möglich, wenn sich das neue Skript an dieselben Ausgaben wie das alte hält. Wie viel das an Aufwand bedeutet, hängt vom jeweiligen Skript ab.

- **"Hübsches Stylesheet":**
  Durch die Verwendung von HTML und CSS ("Stylesheet") könnten die exportierten XML-Dateien auch grafisch aufbereitet werden. Pro XML-Typ wird allerdings ein eigenes Stylesheet benötigt.

- **Weitere Ausgabeformate:**
  Die Ausgabe könnte zusätzlich Vormaten wie beispielsweise als eine Excel-Datei erfolgen. Dies könnte durch ein einzelnes zusätzliches Skript, welches die Datenbank in das jeweilige Format überführt, realisiert werden.

Die erwähnte Änderung der Struktur betrifft das Zusammenführen einzelner Skripte. Geplant ist voraussichtlich Folgendes:

1. Die Auswertung der übergebenen Parameter von den drei einzelnen Skripten in ein neues verlegen.

2. Die URL-Tests für den jeweiligen Browser in ein neues Skript überführen.

3. Die Funktionen der beiden Skripte "generate-work-db-recover" und "generate-work-db-undefined" direkt in "generate-work-db" einpflegen.

Zusätzlich sind aktuell folgende neue Funktionen geplant:

1. Benachrichtigungen beim Programmende in Form von Ton und Mail.
2. Einfügen eines optionalen MD5-Checks für das gesamte Image.
3. Erweiterung des interaktiven Modus bei der Ausführung, wie zum Beispiel eine Abfrage zur Auswahl der Benutzer nach dem Suchvorgang.

Wie daraus ersichtlich, sind noch einige Punkte für das Programm geplant. Daher wird die Entwicklung nicht mit der Abgabe dieser Arbeit eingestellt werden. Die auf der beigelegten DVD vorhandene Version ist, wie bereits mehrfach erwähnt, für die gestellte Anforderung voll funktionstüchtig und kann problemlos mit allen im Zuge der Arbeit beschriebenen Funktionen genutzt werden.

9 Quellenangabe

Literatur

[1] Rohyt-Belani. Web Browser Forensics, Part 2; 5.12.2010. Website. Available from: http://www.symantec.com/connect/de/articles/web-browser-forensics-part-2.

[2] Flock-Inc. Flock, Social Web-Browser; 4.12.2010. Website. Available from: http://flock.com/.

[3] RockMelt-Inc. RockMelt, Social Web-Browser; 4.12.2010. Website. Available from: http://www.rockmelt.com/.

[4] ELinks-Team. ELinks - Full-Featured Text WWW Browser; 5.12.2010. Website. Available from: http://elinks.or.cz/.

[5] Darknet-Team. 10 Best Security Live CD Distros (Pen-Test, Forensics & Recovery); 6.12.2010. Website. Available from: http://www.darknet.org.uk/2006/03/10-best-security-live-cd-distros-pen-test-forensics-recovery/.

[6] Keith-Jones. Web Browser Forensics, Part 1; 5.12.2010. Website. Available from: http://www.symantec.com/connect/de/articles/web-browser-forensics-part-1.

[7] Keith-Jones. ODESSA - The Open Digital Evidence Search and Seizure Architecture is a cross-platform framework for performing Computer Forensics and Incident Response.; 3.12.2010. Website. Available from: http://sourceforge.net/projects/odessa/files/.

[8] MANDIANT-Corporation. Web Historian; 3.12.2010. Website. Available from: http://www.mandiant.com/products/free_software/web_historian/.

[9] Werner-Rumpeltesz. Historian; 6.12.2010. Website. Available from: http://www.gaijin.at/dlhistorian.php.

[10] AccessData-Group. Forensic Toolkit 3; 6.12.2010. Website. Available from: http://www.accessdata.com/forensictoolkit.html.

[11] Brian-Carrier. The Sleuth Kit; 3.12.2010. Website. Available from: http://www.sleuthkit.org/.

[12] Cygwin-Team. GNU + Cygnus + Windows = Cygwin; 5.12.2010. Website. Available from: http://www.cygwin.com/.

[13] Christophe-Grenier. PhotoRec, Digital Picture and File Recovery; 3.12.2010. Website. Available from: http://www.cgsecurity.org/wiki/PhotoRec.

[14] Soft411. Chrome Browser Cache Viewer; 6.12.2010. Website. Available from: http://www.soft411.com/company/NirSoft-Freeware/ChromeCacheView.htm.

[15] Kamkov-Andrey. Browser Cache Index Viewer; 6.12.2010. Website. Available from: http://www.debryansk.ru/~kamkov/.

[16] Andrew-Stuart. Index.Dat Viewer and Zapper; 6.12.2010. Website. Available from: http://www.sudokuwiki.org/indexdat.htm.

[17] Acesoft. Index.dat Viewer; 6.12.2010. Website. Available from: http://www.acesoft.net/index.dat%20viewer/index.dat_viewer.htm.

[18] Cleanersoft. IE History Manager 1.0; 6.12.2010. Website. Available from: http://www.cleanersoft.com/iehistory/iehistory.htm.

[19] Remote-Exploit. BackTrack – Penetration Testing Distribution; 6.12.2010. Website. Available from: http://www.backtrack-linux.org/.

[20] STD-Staff. STD 0.1 - security tools distribution; 6.12.2010. Website. Available from: http://s-t-d.org/download.html.

[21] Inside-Security. Inside Security Rescue Toolkit; 6.12.2010. Website. Available from: http://www.inside-security.de/insert_en.html.

[22] e fense. Helix 3; 6.12.2010. Website. Available from: http://www.e-fense.com/products.php.

[23] Oliver-Sucker. EDV/IT-Forensik - Der Wirtschafts- und Cyberkriminalität auf der Spur.; 6.12.2010. Website. Available from: http://www.forensic-investigations.de/Leistungen/EDV-IT-Forensik.

[24] Mozilla-Foundation. Produkte von Mozilla; 4.12.2010. Website. Available from: http://www.mozilla-europe.org/de/products/.

[25] w3schools Team. Firefox Version Statistics; 4.12.2010. Website. Available from: http://www.w3schools.com/browsers/browsers_firefox.asp.

[26] Opera-Software-ASA. Opera 10.63 Web-Browser; 4.12.2010. Website. Available from: http://www.opera.com/.

[27] Google. Google Chrome; 5.12.2010. Website. Available from: http://www.google.com/chrome/intl/en/more/index.html.

[28] Microsoft. Lernen Sie Internet Explorer 9 kennen.; 6.12.2010. Website. Available from: http://windows.microsoft.com/de-DE/internet-explorer/products/ie-9/home.

[29] Mike-Murr. The Meaning of LEAK Records; 6.12.2010. Website. Available from: http://www.forensicblog.org/2009/09/10/the-meaning-of-leak-records/.

[30] The-FreeBSD-Project. FreeBSD - The Power To Serve; 4.12.2010. Website. Available from: http://www.freebsd.org/.

[31] Oracle-Corporation. OpenSolaris Community; 4.12.2010. Website. Available from: http://hub.opensolaris.org/.

[32] Nexenta-Systems. nexenta.org - Power of OpenSolaris, with the usability of Linux; 3.12.2010. Website. Available from: http://www.nexenta.org/.

[33] Bash-Entwickler. The GNU Bourne-Again SHell; 3.12.2010. Website. Available from: http://tiswww.case.edu/php/chet/bash/bashtop.html.

[34] Klaus-Gerhardt. Bash-Befehle und Bash-Programmierung; 5.12.2010. Website. Available from: http://linuxseiten.kg-it.de/index.php?index=themes_bash.

[35] FSF. The GNU Awk User's Guide; 3.12.2010. Website. Available from: http://www.gnu.org/manual/gawk/gawk.html.

[36] FSF. cat: Concatenate and write files; 3.12.2010. Website. Available from: http://www.gnu.org/software/coreutils/manual/html_node/cat-invocation.html.

[37] FSF. cut: Print selected parts of lines; 3.12.2010. Available from: http://www.gnu.org/software/coreutils/manual/html_node/cut-invocation.html.

[38] FSF. cp: Copy files and directories; 3.12.2010. Website. Available from: http://www.gnu.org/software/coreutils/manual/html_node/cp-invocation.html.

[39] Vincent-Stemen. Website of dialog; 5.12.2010. Website. Available from: http://hightek.org/dialog/.

[40] fdisk Team. Welcome to FDISK.COM.; 3.12.2010. Website. Available from: http://www.fdisk.com/.

[41] FSF. Findutils; 3.12.2010. Website. Available from: http://directory.fsf.org/project/findutils/.

[42] FSF. Website of Grep; 3.12.2010. Website. Available from: http://www.gnu.org/software/grep/.

[43] FSF. tee: Redirect output to multiple files or processes; 3.12.2010. Website. Available from: http://www.gnu.org/software/coreutils/manual/html_node/tee-invocation.html.

[44] Karel-Zak. util-linux-ng ... a fork of util-linux; 6.12.2010. Website. Available from: http://userweb.kernel.org/~kzak/util-linux-ng/.

[45] FSF. ls: List directory contents; 3.12.2010. Website. Available from: http://www.gnu.org/software/coreutils/manual/html_node/ls-invocation.html.

[46] Jesse-Kornblum. md5deep and hashdeep; 3.12.2010. Website. Available from: http://md5deep.sourceforge.net/.

[47] FSF. md5sum: Print or check MD5 digests; 3.12.2010. Website. Available from: http://www.gnu.org/software/coreutils/manual/html_node/md5sum-invocation.html.

[48] FSF. mkdir: Make directories; 3.12.2010. Website. Available from: http://www.gnu.org/software/coreutils/manual/html_node/mkdir-invocation.html.

[49] FSF. mv: Move (rename) files; 3.12.2010. Website. Available from: http://www.gnu.org/software/coreutils/manual/html_node/mv-invocation.html.

[50] FSF. rm: Remove files or directories; 3.12.2010. Website. Available from: http://www.gnu.org/software/coreutils/manual/html_node/rm-invocation.html.

[51] Wayne-Davison. Welcome to the rsync web pages; 3.12.2010. Website. Available from: http://www.samba.org/rsync/.

[52] Paolo-Bonzini. Sed - A stream-oriented non-interactive text editor; 3.12.2010. Website. Available from: http://directory.fsf.org/project/sed/.

[53] SQLite-Consortium. SQLite Webseite; 3.12.2010. Website. Available from: http://www.sqlite.org/.

[54] FSF. touch: Change file timestamps; 3.12.2010. Website. Available from: http://www.gnu.org/software/coreutils/manual/html_node/touch-invocation.html.

[55] Greg-Roelofs. Zip 3.0; 3.12.2010. Website. Available from: http://www.info-zip.org/Zip.html.

[56] FSF. Introduction to Coreutils; 3.12.2010. Website. Available from: http://www.gnu.org/software/coreutils/.

[57] FSF. Announcing ncurses 5.7; 3.12.2010. Website. Available from: http://www.gnu.org/software/ncurses/.

[58] Thierry-Godefroy. Xdialog home page; 5.12.2010. Website. Available from: http://xdialog.dyns.net/.

[59] GTK-Team. The GTK+ Project; 3.12.2010. Website. Available from: http://www.gtk.org/.

[60] Zenity-Entwicklungsteam. Zenity project Website; 5.12.2010. Website. Available from: http://freshmeat.net/projects/zenity.

[61] QEMU-Team. QEMU - open source prozessor emulator; 4.12.2010. Website. Available from: http://wiki.qemu.org/Main_Page.

[62] Google. Verwenden von nicht alphanumerischen Zeichen in XML-Sitemap-URLs; 5.12.2010. Website. Available from: http://www.google.com/support/webmasters/bin/answer.py?hl=de&answer=35653.

[63] aptosid Team. aptosid spezifische Parameter (nur Live-CD); 3.12.2010. Website. Available from: http://manual.aptosid.com/de/cheatcodes-de.htm#cheatcodes.

[64] aptosid Team. Webseite der Linux-Distribution aptosid; 3.12.2010. Website. Available from: http://aptosid.com.

[65] Jaws-Project. Sqliteman - Sqlite Databases Made Easy; 6.12.2010. Website. Available from: http://sqliteman.com/.

[66] Mauricio-Piacentini. SQLite Database Browser; 6.12.2010. Website. Available from: http://sqlitebrowser.sourceforge.net/.

[67] Wine-Projekt. WINE - Führt Windows-Anwendungen unter Linux, BSD, Solaris und Mac OS X aus.; 6.12.2010. Website. Available from: http://www.winehq.org/.

Gerald Prock

Bakk.

Persönliche Informationen

Name	Gerald Prock, Bakk.
Geburtsdatum	1984
Geburtsort	Schwaz in Tirol
Mutter	Andrea Prock, MA Diplom-Pädagogin
Vater	Thomas Moser Angestellter
Sta.	Österreich

Schulische Ausbildung

1991 bis 1995	(4 Jahre) Volksschule in Jenbach
1995 bis 1999	(4 Jahre) Gymnasium in Schwaz
1999 bis 2005	(6 Jahre) HTBLA-Jenbach Zweig: Mechatronik, Abschluss: Matura

Akademische Ausbildung

2005 bis 2008	(3 Jahre) UMIT - Private Universität für Gesundheitswissenschaften, Medizinische Informatik und Technik GmbH - Hall in Tirol Zweig: Biomedizinische Informatik, Abschluss: Bakk.
seit 2008	JKU - Johannes Kepler Universität - Linz Zweig: Netzwerke und Sicherheit

Berufliche Tätigkeiten

2006 bis 2010	Mitarbeit an der Linuxdistribution sidux
seit Nov. 2008	Geringfügig Angestellter im Landeskrankenhaus Innsbruck

Ferialarbeiten

Juli 2002	Lüftungs- und Klimatechnik, Ing. G. Trenkwalder GmbH
Juli 2003	IT-Support-Center, Landeskrankenhaus Innsbruck
Juli 2004	Technisches-Service-Center, Landeskrankenhaus Innsbruck
Juli 2005	Automatisierungstechnik, Amadyne in Bühl (D)
August 2006	Medizintechnik, Landeskrankenhaus Innsbruck
September 2007	Rechenzentrum, Daten-Verarbeitung-Tirol
Juli 2008	Betriebsrat, Landeskrankenhaus Innsbruck
August 2008	Rechenzentrum, Daten-Verarbeitung-Tirol

Qualifikationen

Gutes Wissen:

- Konfiguration und Administration von Computernetzwerken
- Erstellung, Installation und Wartung von GNU|Linux-Systemen
- Installation, Konfiguration und Wartung von Windows-Systemen
- Webdesign mittels CMS-Systemen
- Bildbearbeitung
- Videoschnitt

Basiswissen:

- Planung und Herstellung von Embedded Controller
- Programmierung von Maschinensteuerungen