Forschungsinstitut
Cyber Defence
Universität der Bundeswehr München

Open C3S
Open Competence Center for Cyber Security

Bachelorstudiengang Informatik/IT-Sicherheit

Weiterführende Themen der Computerforensik [AdFor]

Autoren:
Prof. Dr. Harald Baier
Thomas Göbel, M.Sc.

Universität der Bundeswehr München

Weiterführende Themen der Computerforensik [AdFor]

Autoren:

Prof. Dr. Harald Baier

Thomas Göbel, M.Sc.

---

2. Auflage

Universität der Bundeswehr München

Forschungsinstitut Cyber Defence (CODE)
Carl-Wery-Straße 22
81739 München

2. Auflage (2021-10-29)

Autoren der früheren Auflage:
Björn Roos, M.Sc.
Sebastian Gärtner, M.Sc.

Um die Lesbarkeit zu vereinfachen, wird auf die zusätzliche Formulierung der weiblichen Form bei Personenbezeichnungen verzichtet. Wir weisen deshalb darauf hin, dass die Verwendung der männlichen Form explizit als geschlechtsunabhängig verstanden werden soll.

Inhaltsverzeichnis

Einleitung zu den Studienbriefen

I. Abkürzungen der Randsymbole und Farbkodierungen

Beispiel	B
Definition	D
Kontrollaufgabe	K
Merksatz	M
Übung	Ü

II. Zu den Autoren

Harald Baier promovierte 2002 an der TU Darmstadt über eine Arbeit zur effizienten Erzeugung elliptischer Kurven. Er war Mitarbeiter in einem Sicherheitsprojekt der Deutsche Bank AG und baute das Darmstädter Zentrum für IT-Sicherheit auf. Nach Professorentätigkeiten an der TH Bingen (2004-2009) und der Hochschule Darmstadt (2009-2020) ist er seit 01.09.2020 am Forschungsinstitut CODE der Fakultät für Informatik an der Universität der Bundeswehr München tätig. Dort hat er die Professur für digitale Forensik inne. Schwerpunkt seiner Arbeit sind daher unterschiedliche Aspekte der digitalen Forensik.

Thomas Göbel studierte Informationstechnik im Bachelorstudium an der DHBW Mannheim und später Informatik im Masterstudium an der Hochschule Darmstadt. Im Masterstudiengang spezialisierte er sich auf den Vertiefungsschwerpunkt IT-Sicherheit. Aktuell ist er als Doktorand im Bereich der Digitalen Forensik am Forschungsinstitut CODE an der Universität der Bundeswehr München tätig. Im Detail beschäftigt er sich mit der Synthese von forensisch relevanten Datensätzen und geeigneten IT-forensischen Analysemethoden. Seit Sommersemester 2018 ist er zudem als Dozent und Tutor im Open C³S Projekt tätig.

III. Modullehrziele

Die Studierenden sollen folgende Fähigkeiten erlangen:

- Tiefergehendes technisches Wissen forensischer Wissenschaften.
- Ausführliche Kenntnisse zur Dokumentation und Gutachtenerstellung.
- Verständnis des Aufbaus und IT-forensische Analyse der Windows-Registry.
- Kenntnisse gängiger Artefakte im Windows Betriebssystem sowie deren forensische Relevanz.
- Detailliertes technisches Wissen zur Sicherung und Analyse des Windows Hauptspeichers.
- Bewertung und Kategorisierung anti-forensischer Maßnahmen.

Modulbeschreibung

Modulbezeichnung:	Weiterführende Themen der Computerforensik (Advanced topics in digital forensics)
Studiengang:	
Verwendbarkeit:	Wahlpflichtmodul
Lehrveranstaltungen und Lehrformen:	Weiterführende Themen der Computerforensik Selbstlernphasen, Übungen, Online-Phasen, Präsenzveranstaltung
Modulverantwortliche(r):	Prof. Dr. Harald Baier
Lehrende:	Prof. Dr. Harald Baier
Dauer:	1 Semester
Credits:	5 ECTS-Punkte
Studien- und Prüfungsleistungen:	IT-forensisches Gutachten (Hausarbeit) und Präsentation
Berechnung der Modulnote:	keine Angabe
Notwendige Voraussetzungen:	
Empfohlene Voraussetzungen:	• Einführung in die digitale Forensik
Unterrichts- und Prüfungssprache:	Deutsch
Zuordnung des Moduls zu den Fachgebieten des Curriculums:	IT-Sicherheit Vertiefung
Einordnung ins Fachsemester:	Ab Studiensemester 6
Generelle Zielsetzung des Moduls:	
Arbeitsaufwand bzw. Gesamtworkload:	Präsenzstudium: 15 Zeitstunden Fernstudienanteil: 135 Zeitstunden • Selbststudium: 90 Zeitstunden • Aufgaben: 30 Zeitstunden • Online-Betreuung: 15 Zeitstunden **Summe: 150 Zeitstunden**

Lerninhalt und Niveau:	In diesem Modul werden die folgenden Themengebiete behandelt: • Gutachtenerstellung. • Aufbau und forensische Untersuchung der Windows-Registry. • Windows Artefakte. • Möglichkeiten und Techniken der Antiforensik. • Sicherung und forensische Analyse des Hauptspeichers. • Forensische Analyse von SQLite Datenbanken. Das Niveau der Lerninhalte liegt gemessen am DQR-Niveau bei 6 (Bachelor)
Angestrebte Lernergebnisse:	*Fachkompetenz:* Die Studierenden erlernen die Analyse und Auswertung der Grundlegenden forensischen Artefakte innerhalb des Windows Betriebssystems und haben Kenntnisse in der Untersuchung anwendungsspezifischer Daten. Des Weiteren sind die Studierenden mit den weiterführenden Techniken der Hauptspeicherforensik vertraut. Sie kennen weiter gängige anti-forensische Maßnahmen und sind sich deren Auswirkungen auf den Untersuchungsprozess bewusst. Darüber hinaus werden den Studierenden Konzepte für die Erstellung gerichtsverwertbarer Gutachten vermittelt. Zudem erlernen die Studierenden den Aufbau und die forensische Analyse von gängigen SQLite Datenbanken. *Methodenkompetenz:* Die Studierenden beherrschen den Umgang mit den forensischen Tools und können wichtige Ergebnisse daraus eigenständig entnehmen. Sie sind mit weiterführenden Themen und Konzepte der IT-Forensik vertraut und können diese bei einer forensischen Untersuchung anwenden. Sie können weiter mit dem erlangten Wissen aus dem Modul sicher umgehen und können Aufgaben und Problemstellungen nachvollziehen und lösen. *Sozialkompetenz:* Die Studierenden erlernen aufgrund gemeinsamer forensischen Untersuchungen im Team zu arbeiten und können auftretende Probleme, Fragen und Aufgaben durch fachgebundene Diskussion lösen. *Selbstkompetenz:* Die Studierenden erlangen die Fähigkeit, eine forensische Untersuchung durchzuführen und sind in der Lage die Ergebnisse zu bewerten. Weiterhin besitzen Sie die Kompetenz, sich an neue Gegebenheiten anzupassen und können so auf veränderte Hardware und Software reagieren.
Häufigkeit des Angebots:	ca. alle 2 Jahre
Anerkannte Module:	
Anerkannte anderweitige Lernergebnisse / Lernleistungen:	

Medienformen:	Studienbriefe in schriftlicher und elektronischer Form, Onlinematerial in Lernplattform, Übungen und Projekt über Lernplattform, Online-Konferenzen, Chat und Forum, Präsenzveranstaltung mit Rechner und Beamer.
Literatur:	Als begleitende und vertiefende Literatur wird empfohlen: • Eigenes Skript • Michael Hale Ligh, et al.: The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory. John Wiley & Sons, 2014, ISBN 978-1118825099 • Harlan Carvey: Windows Registry Forensics: Advanced Digital Forensic Analysis of the Windows Registry. Elsevier, 2011, ISBN 978-0128032916 • Paul Sanderson, et al.: SQLite Forensics. Independently published, 2018, ISBN 978-1980293071 • Eoghan Casey (Hrsg.): Handbook of computer crime investigation. Forensic tools and technology. 6th Printing. Elsevier Academic Press, Amsterdam u. a. 2007, ISBN 978-0-12-163103-1. • Alexander Geschonneck: Computer-Forensik. Computerstraftaten erkennen, ermitteln, aufklären. 5. aktualisierte und erweiterte Auflage. dpunkt Verlag, Heidelberg 2011, ISBN 978-3-89864-774-8. Weitere Literatur wird in der Lehrveranstaltung bekannt gegeben.

Studienbrief 4 Windows Artefakte

4.1 Lernziele

Dieses Kapitel befasst sich mit weiteren Aspekten der forensischen Analyse des Windows Betriebssystems. Nach der Durcharbeitung dieses Studienbriefs sind Sie mit gängigen Artefakten im Windows Betriebssystem und können darüber hinaus deren forensischen Wert fallbezogen abschätzen.

4.2 Einleitung

Artefakt

Unter einem *Artefakt* versteht man allgemein ein durch menschliches Zutun entstandes Produkt oder Phänomen[1]. Ein Artefakt unterscheidet sich damit von natürlichen oder vom Menschen unbeeinflussten Phänomenen. Die konkrete Bedeutung des Begriffs *Artefakt* hängt noch vom Kontext ab, in dem der Begriff verwendet wird. Im Bereich der Forensik versteht man unter einem Artefakt eine vom Nutzer erzeugte Spur, die typischerweise unbewusst erzeugt wird. Im Fall des Betriebssystems Windows sind daher Windows Artefakte digitalen Spuren, die ein Nutzer durch den Umgang mit dem Betriebssystem erzeugt.

SANS cheat sheet

Das SANS Institut[2] ist eine privatwirtschaftlich orientierte Weiterbildungseinrichtung in den USA. SANS hat sich bereits seit über 25 Jahren auf Fortbildungen im Bereich der Cybersicherheit spezialisiert. SANS hat im Rahmen eines Kurses auf dem Gebiet der *Digital Forensics and Incident Response* (DFIR) eine Übersicht über Artefakte erstellt, die im Zusammenhang mit der IT-forensischen Untersuchung des Windows Betriebssystems von Bedeutung sind[3]. Eine Adaption dieser Übersicht finden Sie in Abbildung 4.1. Ziel von SANS ist es, einen Spickzettel (cheat sheet) bereitzustellen, um Spuren in einem Windows-System nach Aktivitätskategorien zu sortieren und so leichter zu finden.

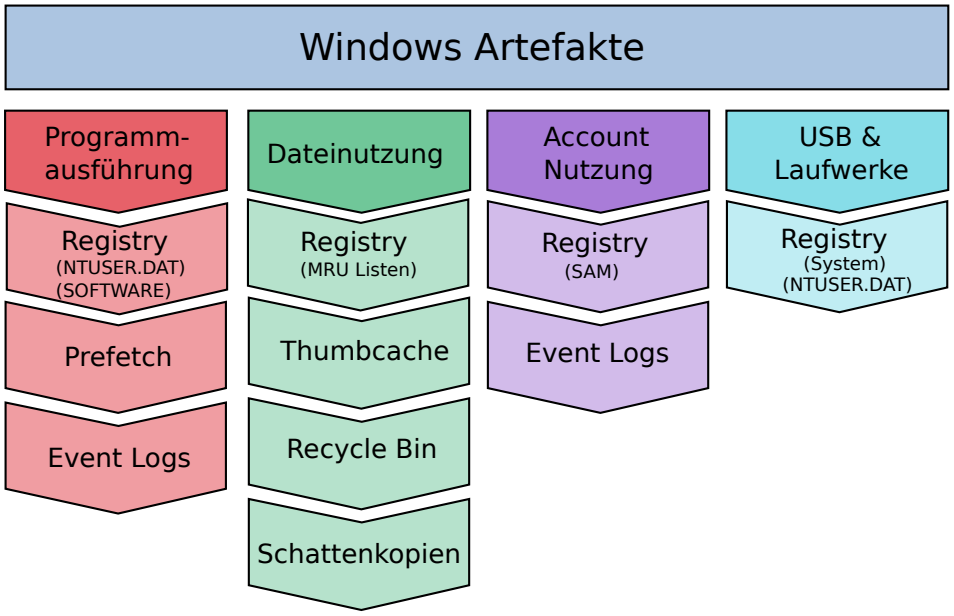


Abb. 4.1: Windows Artefakte in Anlehnung an SANS

Für unsere Betrachtungen sind die in Abbildung 4.1 dargestellten vier Kategorien aus dem SANS cheat sheet von Bedeutung:

[1] https://de.wiktionary.org/wiki/Artefakt, Zugriff am 31.10.2016
[2] http://www.sans.org/
[3] https://uk.sans.org/posters/windows_artifact_analysis.pdf, Zugriff am 22.10.2016

Programmausführung

1. Durch die Ausführung eines Executables erzeugt der Nutzer an zahlreichen Stellen des Windows Betriebssystems Spuren. Von Bedeutung sind insbesondere der Registry Hive des Nutzers aus der Datei `NTUSER.DAT` sowie die globalen Programminformationen aus dem `SOFTWARE`-Hive (die Analyse der Registry wurde in Abschnitt 3.5 vorgestellt). Programmausführungen führen auch dazu, dass das Betriebssystem beliebte Applikationen 'auf Vorrat' in den Arbeitsspeicher lädt, damit der Nutzer diese bei Bedarf schneller nutzen kann. Die Analyse der Spuren, die durch dieses *Prefetching* hervorgerufen werden, beschreiben wir in Abschnitt 4.4. Schließlich führt die Ausführung von Programmen zur Erzeugung von Ereignissen (sogenannte *Events*). Die Analyse von Windows Event Logs wird in diesem Studienbrief nicht weiter thematisiert.

Dateinutzung

2. Die Nutzung einzelner Dateien lässt sich einfach durch die *Most Recently Used* (MRU) Listen der Registry analysieren. Für Bilder hält das Betriebssystem sogenannte *Thumbcaches* vor, die kleine Vorschaubilder für Bilddateien der einzelnen Verzeichnisse effizient bereitstellen. Auf Thumbcaches gehen wir in Abschnitt 4.3 ein. Beim Löschen einer Datei mittels Bordmitteln des Betriebssystems verschiebt Windows diese in den Papierkorb. Spuren zu gelöschten Dateien findet man daher im Verzeichnis `Recycle.bin`. Wir beschreiben die Analyse dieser Spuren in Abschnitt 4.5. *Schattenkopien* oder *Volume Shadow Copies* sind Wiederherstellungspunkte für das Betriebssystem. Falls sich Windows in einem instabilen Zustand befindet (beispielsweise durch falsche Konfiguration oder Treiberinstallation), können mit den Schattenkopien wichtige Systemdateien zu einem bestimmten Zeitpunkt X wiederhergestellt werden. Wir gehen darauf nicht näher ein.

Account Nutzung

3. Spuren zu vorhandenen Accounts sowie deren Nutzung lassen sich über den SAM der Registry (siehe Abschnitt 3.6.2) oder über die Windows Event Logs analysieren.

USB und Laufwerke

4. Informationen zu angeschlossenen Massenspeichern erhält der IT-Forensiker aus der Registry. Zentrale Quellen sind der Registry Hive des Nutzers aus der Datei `NTUSER.DAT` sowie die globalen Informationen aus dem `SYSTEM`-Hive. Wir haben dies für USB-Massenspeicher im Detail in Abschnitt 3.6.3 vorgestellt.

K

Kontrollaufgabe 4.1: Betriebssystem Artefakte

Welche Informationen lassen sich aus der Analyse sogenannter Artefakte gewinnen?

K

Kontrollaufgabe 4.2: Windows Artefakte

Aus welchem Grund eignen sich die im Artefakte des Betriebssystems für eine forensische Analyse?

4.3 Thumbcaches

Thumbcache

Innerhalb von Thumbcaches speichert Windows die Vorschaubilder, die der Windows Explorer zur Symbolansicht anzeigt, wenn ein Nutzer die in einem Ordner abgelegten Dateien betrachtet. Thumbcaches werden aus Usability-Gründen erzeugt, sobald der Nutzer den Inhalt eines Verzeichnisses mittels des Explorers ansieht. Für Bilddateien sieht der Nutzer dann schon im Explorer, welche Bilder in dem Verzeichnis liegen. Die Wortbedeutung setzt sich zusammen aus *Thumb*

(eigentlich Daumen, in der Informatik gebräuchlich für kleine Vorschaubilder) und *Cache* (Zwischenspeicher).

Voraussetzung

Für die Existenz eines Vorschaubildes in den Thumbcache-Dateien muss der Nutzer den Ordner mit Bilddateien im Explorer geöffnet haben. Dies gilt auch für Bilder, die sich auf externen Datenträgern befinden – deren Vorschaubilder werden ebenfalls im Thumbcache gespeichert, falls der Windows Explorer zur Verzeichnisansicht genutzt wurde.

Tools

Zum Betrachten der Thumbcache-Vorschaubilder ist eine entsprechende Software sehr hilfreich. Wir gehen auf spezielle Tools zur Extraktion der Thumbcache-Bilder in Abschnitt 4.3.2 ein. Zusätzlich lassen sich die Vorschaubilder auch mit klassischen Filecarving-Programmen aus den Thumbcache-Datenbanken extrahieren.

Speicherort

Bei älteren Windows-Versionen (Windows NT, Windows XP) existierte für die Vorschaubilder in dem jeweiligen Verzeichnis eine eigene Datei *thumbs.db* – die Thumbcaches wurden also dezentral gespeichert. Mit der Einführung von Windows Vista ist Microsoft zu einer zentralisierten Speicherung für den jeweiligen Nutzer übergegangen, lediglich für Bilddateien, die auf einem eingebundenen Netzlaufwerk gespeichert sind, existiert in diesem Verzeichnis eine Datei mit dem bekannten Namen `thumbs.db`. Seit Windows Vista befinden sich die Nutzer-spezifischen Thumbcache-Dateien unter

```
%UserProfile%\AppData\Local\Microsoft\Windows\Explorer.        (4.1)
```

Forensischer Nutzen

Der zentrale IT-forensische Nutzen der Thumbcaches besteht darin, Aussagen über gelöschte Bilder zu treffen. Vielen Nutzern ist das Konzept der Thumbcaches nicht bewusst, so dass die Thumbcaches beim Löschen von Bildern typischerweise erhalten bleiben. Oft lassen sich daher für gelöschte Bilder zumindest die Vorschaubilder aus dem Thumbcache extrahieren. Der zweite Nutzen besteht in einer Aussage, welche Bilder sich auf externen Datenträgern befunden haben, die mit dem Betriebssystem verbunden waren.

Randbedingungen

Bei der Analyse der Thumbcaches sind einige Randbedingungen zu beachten. Die Erstellung von Vorschaubildern kann vom Nutzer deaktiviert werden. Er kann dies entweder über die Einstellungen des Explorers durchführen oder über die Gruppenrichtlinien (die Gruppenrichtlinie wird konfiguriert über Benutzerkonfiguration → Administrative Vorlagen → Windows-Komponenten → Windows-Explorer → Zwischenspeicherung von Miniaturansichten deaktivieren). Weiterhin kann ein (halbwegs) fachkundiger Nutzer die Thumbcache-Datenbanken sicher löschen, der Speicherort liegt in seinem persönlichen Profilbereich. Zuletzt ist die Zuordnung von Vorschaubildern zu originalen Dateinamen und dem Dateipfad oftmals nicht möglich.

Dateiname

In dem Verzeichnis (4.1) liegen Dateien der Thumbcache-Datenbanken, deren Dateinamen von der Form `thumbcache_NNN.db` sind. Hierin ist `NNN` entweder eine Zahl oder ein String:

- Im Fall einer Zahl gibt `NNN` die Auflösung der in der Datenbank enthaltenen Vorschaubilder an. Wir sehen uns das in Beispiel 4.1 an. Die Zahl `NNN` gibt die Auflösung der Vorschaubilder in Pixel an, z.B. beträgt für `NNN`= 32 die Auflösung 32x32 Pixel.
- Im Fall eines Strings ist die Datei `thumbcache_idx.db` von Bedeutung. Diese ist eine Index-Datei, die den Zugriff auf die einzelnen Vorschaubilder aus den Datenbanken ermöglicht. Es gibt noch den String `NNN=sr`, der Nutzen

dieser Datei ist aber unklar, sie enthält vermutlich keine relevanten Informationen, da sie nur einen Header einer bekannten Datenstruktur enthält.

ThumbnailcacheID

Zur Referenzierung auf das indizierte Bild nutzt Windows eine *ThumbnailcacheID*. Die genaue Berechnung dieser ID ist kompliziert. Im Wesentlichen werden Daten des Bildes gehasht, die eine eindeutige Referenzierung der Bilddatei ermöglichen: GUID des Volumes (in dem die Datei gespeichert wird), MFT-Entry-Number, Dateiendung sowie der letzte schreibende Zugriff. Durch das Hashen wird die Rückrechnung von gegebener ThumbnailcacheID auf die MFT-Entry-Number und damit auf die zugrundeliegende Datei erschwert. Nur falls die übrigen Informationen vorliegen (also insbesondere der Zeitstempel), könnte man mittels eines Brute-Force-Angriffs die MFT-Entry-Number bestimmen.

B

Beispiel 4.1: Thumbcaches

In diesem Beispiel betrachten wir die Thumbcache-Dateien der Nutzerin Eve. Wie schon in vorangegangenen Beispielen ist die zu analysierende Windows-Systempartition unter `/mnt` in das Dateisystem unserer IT-forensischen Workstation eingebunden. Wir lassen uns die Thumbcache-Dateien von Eve mittels des üblichen List-Befehls `ls` anzeigen.

```
$ ls -l /mnt/Users/eve/AppData/Local/Microsoft/Windows/Explorer
[REMOVED]
-rwxrwxrwx 2 root root      24 Feb 17 14:42 thumbcache_1024.db
-rwxrwxrwx 2 root root 2097152 Feb 17 14:42 thumbcache_256.db
-rwxrwxrwx 2 root root 1048576 Mär 10 10:44 thumbcache_32.db
-rwxrwxrwx 2 root root 2097152 Feb 17 14:42 thumbcache_96.db
-rwxrwxrwx 2 root root    6488 Mär 10 12:04 thumbcache_idx.db
-rwxrwxrwx 2 root root      24 Feb 17 14:42 thumbcache_sr.db
```

Zu sehen sind vier Thumbcache-Datenbanken, die Vorschaubilder in unterschiedlichen Auflösungen bereitstellen (Auflösung 32, 96, 256 sowie 1024). Die Dateigrößen der Thumbcache-Datenbanken zeigen, dass die Auflösung 1024 nicht genutzt wird (die Datei `thumbcache_1024.db` besteht vermutlich nur aus dem Datenbank-Header).

Die Datei `thumbcache_idx.db` ist eine Index-Datei, sie enthält für jedes Vorschaubild die ThumbnailcacheID und die Offsets zu den Thumbnails innerhalb der eigentlichen Thumbcache-Datenbanken. Die Datei `thumbcache_sr.db` hat wie schon die Datei `thumbcache_1024.db` eine Größe von 24 Byte und besteht vermutlich nur aus dem Datenbank-Header.

Ablauf einer Thumb-Anfrage

Der Ablauf der Anfrage für ein Vorschaubild durch den Explorer ist in Abbildung 4.2 dargestellt. Auslöser ist, dass der Verzeichnisinhalt mittels des Explorers vom Nutzer aufgerufen wird. Der Explorer fragt dann Informationen zu dem Vorschaubild der Datei bei der *Windows Search Index Datenbank* `Windows.edb` an. Hierbei handelt es sich um eine Windows interne Datenbank im Extensible Storage Format, die für die Indexierung der Dateien im Dateisystem verantwortlich ist, zum Beispiel für eine schnellere Suche im Suchfeld des Windows Explorers. Die Datenbank speichert unter Anderem die ThumnailcacheID von Bildern im Rahmen der Datei-Indexierung. Die Datei `Windows.edb` liegt im Verzeichnis

```
C:\ProgramData\Microsoft\Search\Data\Applications\Windows.
```

Die Datenbank `Windows.edb` liefert die ThumbnailcacheID des angefragten Vorschaubildes zurück. Über einen Hashlookup dieser ID in der Datei `thumbcache_idx.db` erhält das Betriebssystem die Information, an welchem Offset die Bilder in unterschiedlichen Auflösungen in den Thumbcache-Datenbanken `thumbcache_NNN.db` liegen. Damit kann das Betriebssystem auf die Vorschaubilder zugreifen.

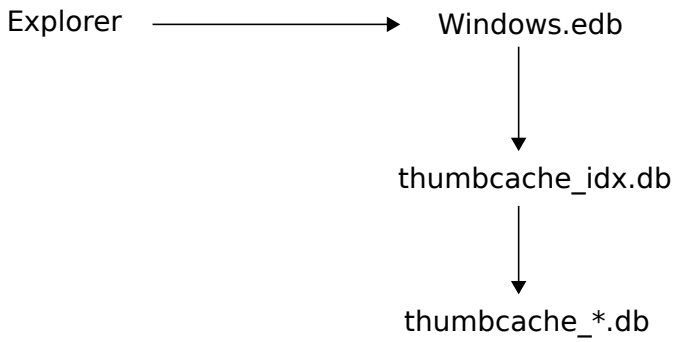


Abb. 4.2: Dateizugriffe bei der Anfrage nach einem Vorschaubild.

Zuordnung Vorschaubild zu Datei

Ein wesentlicher Nachteil des aktuell von Windows verwendeten Datenformats (insbesondere gegenüber Windows XP) ist, dass der Pfad zur originalen Datei bei Vorliegen des Vorschaubildes nicht mehr offensichtlich ist. Die Zuordnung eines Vorschaubilds zu der zugehörigen originalen Datei funktioniert nur über die zentrale *Windows Search Index* Datenbank `Windows.edb`. Allerdings gilt für eine erfolgreiche Zuordnung eines Vorschaubildes zu einem mittlerweile gelöschten Bild die Voraussetzung, dass keine Reorganisation der Indexierungsdatenbank stattgefunden haben darf.

Aktualisierung

Ein Vorschaubild für eine Datei wird in den Thumbcache-Datenbanken aktualisiert, falls das Bild seit dem letzten Anzeigen mittels des Explorers neu angelegt wurde oder das Bild bzw. der Dateipfad verändert wurde.

Kontrollaufgabe 4.3: Thumbcaches

Welchen forensischen Nutzen können Sie aus der Analyse von Thumbcache-Dateien gewinnen?

K

Kontrollaufgabe 4.4: Zuordnung von Vorschaubild zum Original

Lassen sich mittels der Thumbcachedateien sowie zentraler Windows-Datenbanken Informationen zur originalen Bilddatei eines Vorschaubildes auf dem System ermittteln?

K

4.3.1 Datenstrukturen

Das interne Format der Thumbcache-Index-Datei sowie der Thumbcache-Datenbanken ist proprietär. Wie häufig unter Windows sind viele Felder und deren Bedeutung nicht bekannt. Unsere Darstellung orientiert sich an der Arbeit von Joachim Metz [26].

Datenstrukturen der Thumbcache-Index-Datei

Datenstrukturen

Eine Übersicht über die Datenstrukturen einer Thumbcache-Index-Datei finden Sie in Abbildung 4.3. Eine Thumbcache-Index-Datei besteht aus einem Datei-Header

und mehreren Index-Einträgen. Pro Vorschaubild gibt es einen Index-Eintrag, der die Zuordnung von der ThumbnailcacheID zu den Offsets in den Datenbanken ermöglicht. Wir gehen auf die Datenstrukturen in dem Header und den Index-Einträgen im Folgenden ein.

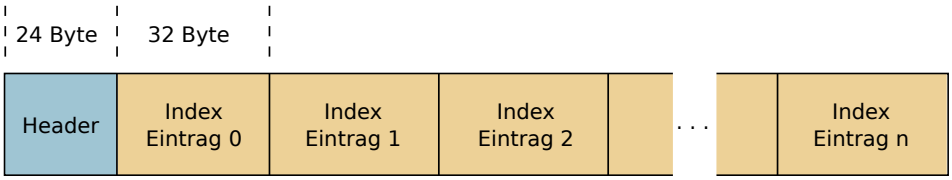


Abb. 4.3: Datenstrukturen der `thumbcache_idx.db`.

Tabelle 4.1: Datenfelder des Headers von `thumbcache_idx.db` ab Windows 7.

Adresse (dez.)	Adresse (hex.)	Beschreibung
00-03	00-03	ASCII-Signatur `IMMM`
04-07	04-07	Version (0x14 = Vista, 0x15 = Windows 7)
08-11	08-0B	Unbekannt
12-15	0C-0F	Anzahl genutzter Einträge
16-19	10-13	Anzahl Einträge
20-23	14-17	Unbekannt

Index-Header

Die Datenfelder des Thumbcache-Index-Headers finden Sie in Tabelle 4.1. Der Header hat eine Länge von 24 Byte, er beginnt mit der ASCII-Signatur `IMMM`. Mittels dieser Signatur könnte man auch gelöschte Index-Dateien ohne Hilfe des Dateisystems finden. Ab Offset 4 steht eine Information über die Windows-Version der Index-Datei. Neben zwei unbekannten Datenfeldern enthält der Index-Header noch Informationen zu Einträgen bzw. genutzten Einträgen in der Index-Datei.

Tabelle 4.2: Datenfelder eines Eintrags von `thumbcache_idx.db` `Eintrags` (Windows 7).

Adresse (dez.)	Adresse (hex.)	Beschreibung
00-07	00-07	`ThumbnailcacheID`
08-11	08-0B	Unbekannt
12-15	0C-0F	Offset in `thumbcache_32.db`
16-19	10-13	Offset in `thumbcache_96.db`
20-23	14-17	Offset in `thumbcache_256.db`
24-27	18-1B	Offset in `thumbcache_1024.db`
28-31	1C-1F	Offset in `thumbcache_sr.db`

Index-Eintrag

Ein Eintrag in der Thumbcache-Index-Datei `thumbcache_idx.db` hat jeweils die Länge 32 Byte. Die Datenfelder eines Eintrags finden Sie in Tabelle 4.2. Die Einträge liegen zwar an nicht-festgelegten Offsets, sie sind aber leicht zu finden, weil die Index-Datei nach dem Header einfach nach Blöcken der Länge 32 Byte durchsucht werden muss – in diesen Blöcken stehen Index-Einträge oder Nullbytes. Die ersten acht Byte eines Index-Eintrags belegt die ThumbnailcacheID. Nach einem unbekannten Datenfeld folgen ab Offset 12 die jeweiligen Adressen der Vorschaubilder unterschiedlicher Auflösung in den zugehörigen Thumbcache-Datenbanken. Ein Offset ist jeweils 4 Byte oder 32 Bit lang. Der Wert `0xFFFFFFFF` wird als negative Zahl -1 interpretiert. Dies bedeutet, dass für das Bild kein Vorschaubild in dieser Auflösung vorliegt. Der letzte Pointer in die Datenbank `thumbcache_sr.db` zeigt auf keine Information, der zugehörige Wert ist `0xFFFFFFFF`.

B

Beispiel 4.2: Thumbcache Index Datei

In diesem Beispiel betrachten wir die Datenstrukturen der Thumbcache-Index-Datei der Nutzerin Eve. Die zu analysierende Windows-Systempartition ist unter `/mnt` in das Dateisystem unserer IT-forensischen Workstation eingebunden. Wir betrachten den Hexdump der Thumbcache-Index-Datei von Eve mittels `xxd`.

```
$ cd /mnt/Users/eve/AppData/Local/Microsoft/Windows/Explorer
$ xxd thumbcache_idx.db

0000000: 494d 4d4d 1500 0000 0400 0000 6900 0000  IMMM........i...
0000010: ca00 0000 0000 0000 0000 0000 0000 0000  ................
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000050: 0000 0000 0000 0000 6a55 9909 0033 2fdd  ........jU...3/.
0000060: 0200 0080 ffff ffff ffff ffff f06a 1500  .............j..
0000070: ffff ffff ffff ffff 63de 4a80 f1a3 12cc  ........c.J.....
[REMOVED]
00002d0: ffff ffff ffff ffff d8f6 bb37 310e d315  ...........71...
00002e0: 0260 0008 ffff ffff ffff ffff 8007 0000  .`..............
00002f0: ffff ffff ffff ffff 09e4 b122 e10c f926  ..........."...&
[REMOVED]
```

In den ersten vier Byte des Headers steht die ASCII-Signatur `IMMM`. Ab Offset 0*x*04 steht die Information, durch welche Windows-Version die Index-Datei angelegt wurde. Der Wert 0x15 zeigt, dass es sich um eine Index-Datei von Windows 7 handelt. Aktuell sind 0x69=105 (Offset 0x0C bis 0x0F) von insgesamt 0xca=202 (Offset 0x10 bis 0x13) Einträgen genutzt. Der Header endet an Offset 0x17=23.

Nach Nullbytes an den Adressen 0x18 bis 0x57 stehen ab Offset 0x58 die Index-Einträge. Ihre Länge ist je 32 Byte, wir haben sie abwechselnd blau und magenta gesetzt. Auf die Datenfelder in einem Index-Eintrag gehen wir in Beispiel 4.3 ein.

B

Beispiel 4.3: Thumbcache Index-Eintrag

Wir betrachten wie schon in Beispiel 4.2 die Datenstrukturen der Thumbcache-Index-Datei der Nutzerin Eve. Im Folgenden sehen wir uns die Datenfelder eines Index-Eintrags am Beispiel des Eintrags ab Offset 0x2d8 der Index-Datei an.

```
$ cd /mnt/Users/eve/AppData/Local/Microsoft/Windows/Explorer
$ xxd thumbcache_idx.db
[REMOVED]
000002d0: ffff ffff ffff ffff d8f6 bb37 310e d315  ...........71...
000002e0: 0260 0008 ffff ffff ffff ffff 8007 0000  .`..............
000002f0: ffff ffff ffff ffff 09e4 b122 e10c f926  ..........."...&
[REMOVED]
```

In den ersten acht Byte eines Eintrags steht die ThumbnailcacheID. Im betrachteten Eintrag ab Offset 0x2d8 lautet dessen Hexdump `d8f6 bb37 310e d315`. Das entspricht dem Wert 0x15d30e3137bbf6d8. Die Bedeutung des 32-Bit-Datenfelds ab Offset 0*x*08 des Eintrags ist unbekannt. Es folgen die Offsets der Vorschaubilder in den einzelnen Thumbcache-Datenbanken,

jeweils kodiert als 32-Bit-Wert. Wir sehen, dass es weder ein Vorschaubild der Auflösung 32 (Hexdump `ffff`) noch ein Vorschaubild der Auflösung 96 (Hexdump `ffff`) gibt. Das Vorschaubild der Auflösung 256 steht ab Offset 0x780=1920 der zugehörigen Thumbcache-Datenbank `thumbcache_256.db`. Ein Vorschaubild der Auflösung 1024 existiert ebenfalls nicht (Hexdump `ffff`).

Datenstrukturen der Thumbcache-Datenbanken

Datenbank-Datei

Die Vorschaubilder befinden sich in den Thumbcache-Datenbank-Dateien. Pro Auflösung gibt es eine solche Datei. Wie üblich besteht eine Datenbank-Datei aus einem Datei-Header und den Thumbcache-Einträgen – diese Einträge nennen wir im Folgenden kurz *Chache-Einträge*. Je ein Vorschaubild wird in einem Cache-Eintrag gespeichert. Zwar geht die Auflösung aus dem Dateinamen der Thumbcache-Datenbank hervor, die Auflösung wird aber durch ein Header-Datenfeld festgelegt.

Tabelle 4.3: Datenfelder des Headers einer `thumbcache_NNN.db`

Adresse (dez.)	Adresse (hex.)	Beschreibung
00-03	00-03	ASCII-Signatur `CMMM`
04-07	04-07	Version (0x14 = Vista, 0x15 = Windows 7)
08-11	08-0B	Auflösung: 0 ≙ 32 , 1 ≙ 96, 2 ≙ 256, 3 ≙ 1024
12-15	0C-0F	Unbekannt (Header Size?)
16-19	10-13	Unbekannt (Offset zum letzten Eintrag?)
20-23	14-17	Unbekannt (Anzahl Einträge?)

Datenbank-Header

Die Datenfelder des Thumbcache-Datenbank-Headers finden Sie in Tabelle 4.3. Der Header besteht aus sechs Datenfeldern zu je vier Byte und hat daher eine Länge von 24 Byte. Er beginnt mit der ASCII-Signatur `CMMM`. Wie Sie gleich sehen werden, beginnen allerdings auch die einzelnen Einträge in der Thumbcache-Datenbank mit dieser ASCII-Signatur – daher ist die Signatur zwar behilflich, gelöschte Thumbcache-Datenbank-Dateien ohne Hilfe des Dateisystems zu finden, allerdings muss man diese Randbedingung beachten. Ab Offset 4 steht eine Information über die Windows-Version der Thumbcache-Datei. Im Datenfeld ab Offset 8 ist die Auflösung kodiert – aktuell gibt es vier Auflösungen, die durch die Zahlen 0 bis 3 kodiert sind. Die tatsächliche Bedeutung der verbleibenden drei Datenfelder ist nicht bekannt, es könnte sich um Angaben zur Header-Größe, Offset zum letzten Eintrag sowie Anzahl der Cache-Einträge. Einen exemplarischen Header finden Sie in Beispiel 4.4.

B

Beispiel 4.4: Header einer Thumbcache-Datenbank-Datei

In diesem Beispiel betrachten wir die Datenfelder des Headers einer Thumbcache-Datenbank-Datei. Besitzerin ist die Benutzerin Eve, die vermutliche Auflösung ist 256. Die zu analysierende Windows-Systempartition ist unter `/mnt` in das Dateisystem unserer IT-forensischen Workstation eingebunden.

```
$ cd /mnt/Users/eve/AppData/Local/Microsoft/Windows/Explorer
$ xxd thumbcache_256.db

0000000: 434d 4d4d 1500 0000 0200 0000 1800 0000  CMMM............
0000010: f4b4 1600 2601 0000 434d 4d4d 8000 0000  ....&...CMMM....
[REMOVED]
016b4f0: 8c99 ffd9 434d 4d4d 0c4b 0900 0000 0000  ....CMMM.K......
016b500: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

```
016b510: 0000 0000 0000 0000 0000 0000 97f0 2514  ..............%.
016b520: 1c2c 3b24 0000 0000 0000 0000 0000 0000  .,;$............
016b530: 0000 0000 0000 0000 0000 0000 0000 0000  ................
016b540: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

In den ersten vier Byte des Headers steht die ASCII-Signatur `CMMM`. Ab Offset 0*x*04 steht die Information, durch welche Windows-Version die Cache-Datei angelegt wurde. Der Wert 0x15 zeigt, dass es sich um eine Cache-Datei von Windows 7 handelt. Das Datenfeld zur Auflösung steht ab Offset 8, der kodierte Wert ist 0x02, es handelt sich also tatsächlich um die Auflösung 256.

Die Inhalte der übrigen Datenfelder ist nicht gesichert. Ab Offset 12 steht der Wert 0x18, der der Header-Größe 24 entspricht. Ab Offset 16 steht der Wert **0x16b4f4**, der möglicherweise den Pointer auf den letzten Cache-Eintrag enthält. Tatsächlich finden sich daher vermutlich keine weiteren Einträge, wie wir den zahlreichen Nullbytes entnehmen können. Ab Offset 20 steht der Wert 0x0126, der möglicherweise die Anzahl der Cache-Einträge angibt. In unserem Beispiel wären dies 294 Cache-Einträge.

Tabelle 4.4: Datenfelder eines Cache-Eintrags in `thumbcache_NNN.db` ab Windows 7

Adresse (dez.)	Größe (Byte)	Beschreibung
00-04	4	ASCII-Signatur `CMMM`
04-07	4	Größe des Eintrags
08-15	8	ThumbnailcacheID
16-19	4	Länge des Identifier Strings in Byte (Wert sei mit l_i bezeichnet)
20-23	4	Anzahl der Padding Bytes (Wert sei mit l_p bezeichnet)
24-27	4	Länge des Vorschaubildes in Byte (Wert sei mit l_b bezeichnet)
28-31	4	Unbekannt
32-39	8	Prüfsumme des Vorschaubildes
40-47	8	Prüfsumme des Headers
ab 48	l_i	Identifier UTF-16
ab 48+l_i	l_p	Padding
ab 48+l_i + l_p	l_b	Vorschaubild

Cache-Eintrag

Ein Cache-Eintrag einer Thumbcache-Datenbank-Datei `thumbcache_NNN.db` hat eine variable Länge. Die ersten 48 Byte sind Datenfelder statischer Länge. Typischerweise ist nur das Vorschaubild von flexibler Länge. Die Datenfelder eines Cache-Eintrags finden Sie in Tabelle 4.4. Ein Cache-Eintrag beginnt mit der gleichen ASCII-Signatur `CMMM` wie die Datei. Damit ist der Beginn eines Cache-Eintrags innerhalb der Thumbcache-Datenbank leicht zu lokalisieren. Ab Offset 4 steht die Größe des Cache-Eintrags, die als 32-Bit-Wert dargestellt wird. Es folgt ab Offset 8 die ThumbnailcacheID des Vorschaubildes. Ab Offset 16 stehen drei je 32-Bit-Werte für Längen:

Länge l_i

- Zunächst finden Sie ab Offset 16 die Länge des Identifier Strings in Byte, die wir mit l_i bezeichnen (der Identifier selbst steht ab Offset 48). Der Identifier ist die UTF-16-Darstellung der Hex-Zeichen der ThumbnailcacheID. Da die ID 8 Byte belegt und damit durch 16 Hex-Zeichen dargestellt wird (und jedes Hex-Zeichen wegen der UTF-16-Kodierung zwei Byte belegt), werden Sie $l_i = 0x20 = 32$ vorfinden.

Länge l_p

- Ab Offset 20 steht die Länge des Paddings zwischen Identifier und Vorschaubild – oft wird kein Padding genutzt, d.h. der entsprechende Längenwert des Paddings ist Null. Die Länge des Paddings bezeichnen wir mit l_p, meist gilt also $l_p = 0$.

Länge l_b

- Ab Offset 24 steht dann die Länge des Vorschaubildes, die wir mit l_b bezeichnen.

Nach zwei Prüfsummen für Vorschaubild und Header steht ab Offset 48 der Identifier im UTF-16-Format, nach einem möglichen Padding folgt das Vorschaubild.

B

Beispiel 4.5: Cache-Eintrag einer Thumbcache-Datenbank-Datei

In diesem Beispiel betrachten wir die Datenfelder eines Cache-Eintrags der Thumbcache-Datenbank-Datei aus Beispiel 4.4. Besitzerin ist die Benutzerin Eve, die Auflösung ist 256. In Beispiel 4.3 haben wir mittels der Index-Datei für die ThumbnailcacheID mit Hexdump `d8f6 bb37 310e d315` festgestellt, dass lediglich ein Vorschaubild der Auflösung 256 existiert und dass dieses innerhalb der Datei `thumbcache_256.db` ab Offset 0x780=1920 steht. Diesen Cache-Eintrag sehen wir uns an.

```
$ cd /mnt/Users/eve/AppData/Local/Microsoft/Windows/Explorer
$ dd if=thumbcache_256.db bs=1 skip=1920 | xxd

[REMOVED]
00000000: 434d 4d4d 9d44 0000 d8f6 bb37 310e d315  CMMM.D.....71...
00000010: 2000 0000 0000 0000 4d44 0000 0000 0000   .......MD......
00000020: 1b7b b178 3b65 8496 aa9f f218 17ae 8b58  .{.x;e.........X
00000030: 3100 3500 6400 3300 3000 6500 3300 3100  1.5.d.3.0.e.3.1.
00000040: 3300 3700 6200 6200 6600 3600 6400 3800  3.7.b.b.f.6.d.8.
00000050: ffd8 ffe0 0010 4a46 4946 0001 0101 0000  ......JFIF......
00000060: 0000 0000 ffdb 0043 0005 0304 0404 0305  .......C........
00000070: 0404 0405 0505 0607 0c08 0707 0707 0f0b  ................
[REMOVED]
00004450: 0574 fb6f 66f9 594f 0d0a def1 8fa6 c86d  .t.of.YO.......m
00004460: 660e 503a 83ca 9ef5 e83e 1b9b 4ebf 8945  f.P:.....>..N..E
00004470: bc83 ccee 8dc3 0f6c 571b 2d93 2313 8e31  .......lW.-.#..1
00004480: 9a86 c637 694b 45b8 10d9 0476 ac2b c235  ...7iKE....v.+.5
00004490: e37b d8da 9539 534a 09e8 7fff d943 4d4d  .....9SJ.....CMM
```

In den ersten vier Byte des Cache-Eintrags steht die ASCII-Signatur `CMMM`. Ab Offset $0x04$ steht die Größe des Cache-Eintrags, hier also der Wert 0x449d=17565. Wir sehen, dass ab Byte 0x449d die ersten drei Byte `CMM` der ASCII-Signatur stehen und dort der nächste Cache-Eintrag beginnt. Die ThumbnailcacheID steht ab Offset 8, es ist der genannte Hexdump `d8f6 bb37 310e d315`. Wie erwartet gilt $\mathbf{l_i} = \mathbf{32}$ (siehe Offset 16), ab Offset 48 sehen Sie den Identifier – es ist die ThumbnailcacheID, wie Sie einfach durch Little-Endian Interpretation des Hexdumps der ID ab Offset 8 sehen. Es gibt kein Padding, da ab Offset 20 der Wert Null kodiert ist.

Die Länge des Payloads steht ab Offset 24, der Wert ist $l_b = 0x444d = 17485$ und damit um $0x50$ kleiner als die Größe des Eintrags. Nach den zwei Prüfsummen ab Offset 32 sowie dem 32 Byte langen Identifier folgt ab Offset 0x50 das Vorschaubild, das mit dem JPEG-Header ffd8 beginnt und in Byte 0x449c mit dem JPG-Footer ffd9 endet.

Extraktion eines Vorschaubildes

Wir extrahieren das Vorschaubild des in Beispiel 4.4 betrachteten Cache-Eintrags. Bekannt ist, dass der Cache-Eintrag ab Offset 0x780=1920 beginnt und dass das Bild ab Offset 0x50=80 relativ zum Beginn des Cache-Eintrags startet. Daher müssen wir relativ zum Beginn der Datenbank-Datei zu Offset 2000 springen und $l_b = 17485$ Byte auslesen. Die Extraktion nehmen wir mittels `dd` vor (wir vergewissern uns zunächst, dass wir im richtigen Verzeichnis sind):

```
$ pwd
/mnt/Users/eve/AppData/Local/Microsoft/Windows/Explorer

$ dd if=thumbcache_256.db of=d8f6bb37310ed315 skip=2000 bs=1 count=17485

$ file d8f6bb37310ed315
d8f6bb37310ed315: JPEG image data, JFIF standard 1.01
```

Der Aufruf von `file` zeigt, dass die Extraktion vermutlich funktioniert hat, das Ergebnis der Extraktion sehen Sie in Abbildung 4.4 Durch das Vorhandensein

Abb. 4.4: Extrahiertes Vorschaubild

von JPG-Header und -Footer ist eine Extraktion mittels File-Carving möglich. Es gibt komfortablere Tools zur Extraktion von Vorschaubildern, auf die wir nun eingehen.

4.3.2 Tools

Im folgenden kurzen Abschnitt wird Ihnen mit dem Thumbcache Viewer ein OpenSource Programm vorgestellt, das auf die Analyse und Extraktion von Windows Thumbcache Dateien spezialisiert ist. Daneben erwähnen wir kurz den EseDataBase Viewer zur Analyse der zentralen *Windows Search Index* Datenbank `Windows.edb`.

Thumbcache Viewer

Thumbcache Viewer

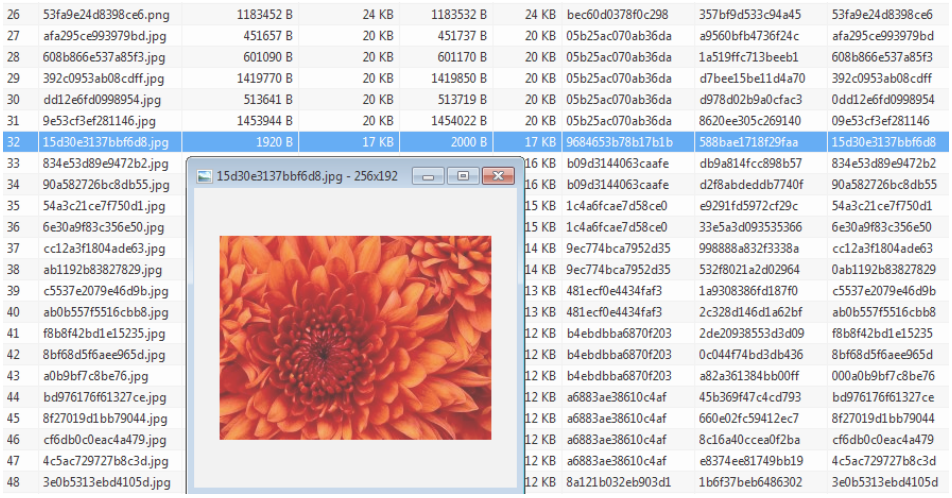
Mit dem Thumbcache Viewer[4] steht eine Software für die Interpretation der Thumbcache-Dateien zur Verfügung. Der Thumbcache Viewer extrahiert die Datenstrukturen aus den Thumbcache-Datenbanken und stellt sie in Tabellenform dar. Die Vorschaubilder können über die Graphische Oberfläche angezeigt und exportiert werden.

Beispieleintrag

Abbildung 4.5 zeigt einen Ausschnitt des Thumbcache Viewers für die Datei aus den Beispielen 4.4 sowie 4.5. Nach einer laufenden Nummer (für die uns interessierende Datei ist das die Nummer 32) folgt ein Dateiname, der gleich der ThumbnailcacheID des Vorschaubildes ist. Die dritte Spalte gibt das Offset des Cache-Eintrages in der zugehörigen Datenbank an, gefolgt von der Größe des Cache-Eintrags. In Spalte 5 finden Sie das Offset zum Vorschaubild im Cache-Eintrag, danach in Spalte 6 die Größe des Vorschaubildes. Es schließen sich drei

[4] https://thumbcacheviewer.github.io/

Abb. 4.5: Extrahierte Informationen des Thumbcache Viewers



26	53fa9e24d8398ce6.png	1183452 B	24 KB	1183532 B	24 KB	bec60d0378f0c298	357bf9d533c94a45	53fa9e24d8398ce6
27	afa295ce993979bd.jpg	451657 B	20 KB	451737 B	20 KB	05b25ac070ab36da	a9560bfb4736f24c	afa295ce993979bd
28	608b866e537a85f3.jpg	601090 B	20 KB	601170 B	20 KB	05b25ac070ab36da	1a519ffc713beeb1	608b866e537a85f3
29	392c0953ab08cdff.jpg	1419770 B	20 KB	1419850 B	20 KB	05b25ac070ab36da	d7bee15be11d4a70	392c0953ab08cdff
30	dd12e6fd0998954.jpg	513641 B	20 KB	513719 B	20 KB	05b25ac070ab36da	d978d02b9a0cfac3	0dd12e6fd0998954
31	9e53cf3ef281146.jpg	1453944 B	20 KB	1454022 B	20 KB	05b25ac070ab36da	8620ee305c269140	09e53cf3ef281146
32	15d30e3137bbf6d8.jpg	1920 B	17 KB	2000 B	17 KB	9684653b78b17b1b	588bae1718f29faa	15d30e3137bbf6d8
33	834e53d89e9472b2.jpg				16 KB	b09d3144063caafe	db9a814fcc898b57	834e53d89e9472b2
34	90a582726bc8db55.jpg				16 KB	b09d3144063caafe	d2f8abdeddb7740f	90a582726bc8db55
35	54a3c21ce7f750d1.jpg				15 KB	1c4a6fcae7d58ce0	e9291fd5972cf29c	54a3c21ce7f750d1
36	6e30a9f83c356e50.jpg				15 KB	1c4a6fcae7d58ce0	33e5a3d093535366	6e30a9f83c356e50
37	cc12a3f1804ade63.jpg				14 KB	9ec774bca7952d35	998888a832f3338a	cc12a3f1804ade63
38	ab1192b83827829.jpg				14 KB	9ec774bca7952d35	532f8021a2d02964	0ab1192b83827829
39	c5537e2079e46d9b.jpg				13 KB	481ecf0e4434faf3	1a9308386fd187f0	c5537e2079e46d9b
40	ab0b557f5516cbb8.jpg				13 KB	481ecf0e4434faf3	2c328d146d1a62bf	ab0b557f5516cbb8
41	f8b8f42bd1e15235.jpg				12 KB	b4ebdbba6870f203	2de20938553d3d09	f8b8f42bd1e15235
42	8bf68d5f6aee965d.jpg				12 KB	b4ebdbba6870f203	0c044f74bd3db436	8bf68d5f6aee965d
43	a0b9bf7c8be76.jpg				12 KB	b4ebdbba6870f203	a82a361384bb00ff	000a0b9bf7c8be76
44	bd976176f61327ce.jpg				12 KB	a6883ae38610c4af	45b369f47c4cd793	bd976176f61327ce
45	8f27019d1bb79044.jpg				12 KB	a6883ae38610c4af	660e02fc59412ec7	8f27019d1bb79044
46	cf6db0c0eac4a479.jpg				12 KB	a6883ae38610c4af	8c16a40ccea0f2ba	cf6db0c0eac4a479
47	4c5ac729727b8c3d.jpg				12 KB	a6883ae38610c4af	e8374ee81749bb19	4c5ac729727b8c3d
48	3e0b5313ebd4105d.jpg				12 KB	8a121b032eb903d1	1b6f37beb6486302	3e0b5313ebd4105d

Prüfsummen an (Daten, Header, Cache-Eintrag) sowie weitere Informationen. Für diesen markierten Eintrag reproduziert der Thumbcache Viewer unsere Ergebnisse aus Beispiel 4.5. Ebenso stellen wir das extrahierte Vorschaubild dar.

Wiederherstellung Dateipfad

Eine weitere Besonderheit der Software ist das automatisierte Mapping der Vorschaubilder zum ursprünglichen Dateipfad. Hierfür muss die Datenbank des Windows Suchdienstes `Windows.edb` zusätzlich eingebunden werden.

EseDatabaseView

Falls Sie das Mapping der Dateipfade manuell vornehmen möchten oder eine Anwendung für das Betrachten der Daten innerhalb der *Windows Search Index Datenbank* `Windows.edb` suchen, können Sie das Programm EseDatabaseView von NirSoft[5] verwenden. Die Daten der Indexierung finden Sie in der Tabelle SystemIndex_0A.

4.4 Prefetch Dateien

Prefetch-Dateien

Prefetch-Dateien werden vom Betriebssystem Windows für häufig ausgeführte Programme angelegt. Hintergrund ist, dass durch die Prefetch-Datei dynamische Bibliotheken und Hilfsdateien frühzeitig geladen werden und damit der Start des Programms schneller gelingt. Zusätzlich zur Optimierung der Programmausführung wird ebenfalls der Windows-Systemstart optimiert. Prefetch-Dateien gibt es seit Windows XP, ihre Dateiendung lautet `*.pf`. Windows legt bis zu 128 Prefetch-Dateien im Verzeichnis `%SystemRoot%\Prefetch` an. Die Datenstrukturen und zentrale Informationen dieses Abschnitts stammen aus dem ForensicsWiki [10] sowie aus der Dokumentation der Bibliothek `libscca` von GitHub [18].

Forensischer Nutzen

Für den IT-Forensiker sind Prefetch-Dateien aus unterschiedlichen Gründen interessant:

- *Ausführung*: Zunächst gibt eine Prefetch-Datei Aufschluss darüber, dass ein Programm ausgeführt wurde. Denn andernfalls hätte das Betriebssystem keine zugehörige Prefetch-Datei angelegt.
- *Häufigkeit der Nutzung*: In der Prefetch-Datei gibt es ein Datenfeld, das die Häufigkeit der Nutzung des Programms zählt. Diese Datenstruktur heißt

[5] http://www.nirsoft.net/utils/ese_database_view.html

Execution Counter oder *Run Count*. Für Windows 7 steht der Execution Counter an Offset $0x98$ (siehe auch Tabelle 4.6), für Windows XP an Offset $0x90$.

- *Zeitstempel*: In der Prefetch-Datei gibt es ein Datenfeld, das den Zeitpunkt der letzten Programmausführung speichert. Für Windows 7 steht dieser Zeitstempel an Offset $0x80$ (siehe auch Tabelle 4.6), für Windows XP an Offset $0x78$. Weiterhin gibt der im NTFS-Dateisystem gespeicherte *created*-Zeitstempel der Prefetch-Datei den Zeitpunkt des Anlegens der Prefetch-Datei an. Sind beide Zeitstempel verschieden, kann eine Aussage über zwei Nutzungszeitpunkte des Executables getroffen werden.
- *Timeline*: Gelingt eine Wiederherstellung gelöschter Prefetch-Dateien, kann der IT-Forensiker mittels der darin gespeicherten Ausführungszeitpunkte sowie der beiden eben genannten Zeitstempel eine Timeline über die Nutzung eines Programms erstellen.
- *Dateinamen*: Den Namen des ausgeführten Programms sowie die geladenen Bibliotheken kann man der Prefetch-Datei entnehmen. Des weiteren kann man Aussagen über den Pfad des Executables treffen. Ggfls. kann der IT-Forensiker mit diesen Informationen nachvollziehen, ob eine veränderte PE-Datei oder eine manipulierte DLL geladen wurde.
- *Persistenz*: Prefetch-Dateien bleiben auch nach der Deinstallation der Anwendung erhalten, so dass Spuren über installierte Programme aus dem Prefetch-Ordner bzw. aus gelöschten Prefetch-Dateien gefunden werden können.

Registry-Einstellung

Die Nutzung von Prefetch-Dateien muss in der Windows-Registry eingestellt werden. Dazu muss der Unterschlüssel `EnablePrefetcher` im Registry-Key

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet00[X]\Control\Session Manager\
Memory Management\PrefetchParameters
```

auf einen Wert ungleich Null gesetzt werden[6]. Typischerweise nutzt man den Wert 3 für den Unterschlüssel `EnablePrefetcher`, um sowohl Prefetching beim Booten als auch Applikations-Prefetching zu aktivieren. Bei der Installation von Windows auf einer SSD (Solid State Drive) ist `EnablePrefetcher` standardmäßig auf Null gesetzt, d.h. es werden per Default keine Prefetch-Dateien erstellt[7].

Dateiname

Der Dateiname einer Prefetch-Datei setzt sich aus dem Dateinamen der ausführbaren Datei und einem 32-Bit-Wert zusammen:

```
<Dateiname des Executables>-<Prefetch-Hash>.pf
```

Der Dateiname der ausführbaren Datei wird dabei in Großbuchstaben geschrieben, der 32-Bit-Wert ergibt sich aus einer 'Prefetch-Hashfunktion', die oft *SCCA*-Hashfunktion genannt wird (der String `SCCA` steht in der Prefetch-Datei als ASCII-Signatur, daher die Bezeichnung der Prefetch-Hashfunktion). Die Prefetch-Hashfunktion ist für die verschiedenen Windows-Versionen unterschiedlich. Die SCCA-Hashfunktion erhält als Eingabe den vollen Dateipfad samt Dateinamen der ausführbaren Datei und gibt einen 32-Bit-Wert aus [18]. Mittels des Prefetch-Hashwerts werden daher Executables mit verschiedenen Pfaden unterschieden. Beispiel 4.6 zeigt einige Prefetch-Dateien einer Windows-Systempartition. In dem

[6] https://msdn.microsoft.com/en-us/library/ms940847(v=winembedded.5).aspx, Zugriff am 02.08.2016

[7] Lt. Vortrag von Alexander Geschonneck über Windows 7 Artefakte beim Anwendertag IT-Forensik am Fraunhofer SIT, 12.04.2011

Beispiel sind zwei verschiedene Prefetch-Dateien für den Mozilla Firefox vorhanden.

B

Beispiel 4.6: Prefetch-Dateien

Wir verwenden eine IT-forensische Workstation mit dem Betriebssystem Linux und untersuchen eine Windows-Systempartition. Die Windows-Partition ist unter `/mnt` in das Dateisystem unserer IT-forensischen Workstation eingehängt. Die Prefetch-Dateien liegen daher im Verzeichnis `/mnt/Windows/Prefetch`, wir zeigen auszugsweise einige Prefetch-Dateien.

```
$ ls -l /mnt/Windows/Prefetch total 9073
[REMOVED]
-rwxrwxrwx 2 root root  39026 Mär 15 15:17 FIREFOX.EXE-7DFC8E4F.pf
-rwxrwxrwx 2 root root 187388 Mär 17 15:16 FIREFOX.EXE-A606B53C.pf
[REMOVED]
-rwxrwxrwx 2 root root 111988 Mar 15 15:17 TOR.EXE-7C6B6C7F.pf
[REMOVED]
-rwxrwxrwx 2 root root 28706 Mär 10 18:44 VERACRYPT.EXE-047EFDD6.pf
-rwxrwxrwx 2 root root 25734 Feb 17 19:54 VERACRYPT FORMAT.EXE-84A1CCFB.pf
[REMOVED]
```

Sie sehen zwei Prefetch-Dateien für den Firefox-Webbrowser. Die jeweiligen Prefetch-Hashwerte sind 7DFC8E4F bzw. A606B53C. Also wurden die jeweiligen Browser-Executables aus unterschiedlichen Pfaden aufgerufen. Des weiteren sehen Sie, dass der Nutzer mindestens einmal die Anonymisierungssoftware Tor bzw. das Verschlüsselungsprogramm `veracrypt` genutzt hat.

Aufbau einer Prefetch-Datei

Der Aufbau einer Prefetch-Datei gliedert sich in einen Prefetch-Header, einen Abschnitt mit weiteren Informationen über den Dateiaufbau (Prefetch File Information) sowie Sektionen A bis D. Bis auf den Prefetch-Header unterscheiden sich die übrigen Abschnitte einer Prefetch-Datei für die unterschiedlichen Windows-Varianten. Wir gehen auf den Aufbau einer Prefetch-Datei im Folgenden ein.

Abb. 4.6: Übersicht über Aufbau einer pf-Datei

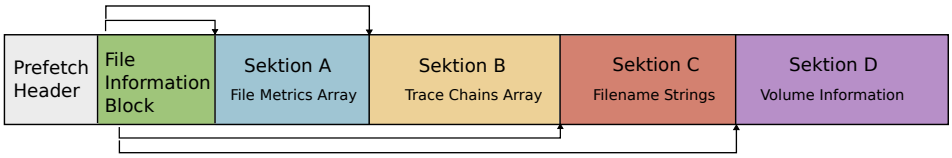


Tabelle 4.5: Datenfelder des Prefetch-Headers

Adresse (dez.)	Adresse (hex.)	Beschreibung
00-03	00-03	Format / Version: $0x17$ für Vista bzw. W7, $0x11$ für XP, $0x1A$ für W8.1, $0x1E$ für W10
04-07	04-07	ASCII-Signatur: `SCCA`
08-11	08-0B	Unbekannt
12-15	0C-0F	Dateigröße der Prefetch-Datei in Byte
16-75	10-4B	Name der ausgeführten Datei (in UTF-16)
76-79	4C-4F	Prefetch Hash (diese 32-Bit-Zeichenkette ist Teil des Dateinamens der Prefetch Datei)

Prefetch File Header

In den ersten 80 Byte einer Prefetch-Datei steht der *Prefetch File Header*. Details zu den Datenfeldern des Prefetch File Headers finden Sie in Tabelle 4.5. Zu Beginn

des Headers stehen Informationen über die Prefetch-Version, die von bestimmten Windows-Versionen genutzt wird. Es folgt die 4-Zeichen lange ASCII-Signatur `SCCA`, die nützlich für das Auffinden einer Prefetch-Datei mittels File Carving ist. Nach einem unbekannten Datenfeld folgt ab Offset 12 die Dateigröße der Prefetch-Datei, die als 32-Bit-Wert in der üblichen Little-Endian-Kodierung abgelegt wird. Ab Offset 16 folgt der Name der ausgeführten Datei und anschließend ab Offset 76 der Prefetch-Hash (zur Berechnung siehe Seite 81). Der Prefetch-Hash ist auch Teil des Dateinamens der Prefetch-Datei, in dem Datenfeld ab Offset 76 ist der Prefetch-Hash Little-Endian gespeichert. In Beispiel 4.7 betrachten wir einen konkreten Prefetch File Header.

B

Beispiel 4.7: Prefetch File Header

Wir verwenden wie in Beispiel 4.6 eine IT-forensische Workstation mit dem Betriebssystem Linux und untersuchen die Prefetch-Datei der Verschlüsselungssoftware `veracrypt` im Verzeichnis `/mnt/Windows/Prefetch`.

```
$ xxd /mnt/Windows/Prefetch/VERACRYPT.EXE-047EFDD6.pf

0000000: 1700 0000 5343 4341 1100 0000 2270 0000  ....SCCA....."p..
0000010: 5600 4500 5200 4100 4300 5200 5900 5000  V.E.R.A.C.R.Y.P.
0000020: 5400 2e00 4500 5800 4500 0000 c0bc f980  T...E.X.E.......
0000030: bbd6 a382 0100 0000 8887 2f86 1100 0000  ........../.....
0000040: 8887 2f86 406d ec85 0000 0000 d6fd 7e04  ../.@m........~.
```

Offset 0 bis 3 entnehmen wir die Windows-Version. Der Wert lautet **0x17**, es handelt sich also um eine Prefetch-Datei von Windows Vista oder Windows 7. Es folgt an Offset 4 bis 7 die ASCII-Signatur `SCCA`. Offset 12 bis 15 entnehmen wir die Dateigröße der Prefetch-Datei, sie beträgt 0x7022=28706 Byte. Das ist konsistent mit der Dateigröße im Dateisystem, wie wir sie in Beispiel 4.6 sehen.

Ab Offset $16 = 0x10$ folgt der Name der ausgeführten Datei, wie unter Windows üblich ist dieser in UTF-16 kodiert. Wir sehen, dass der Dateiname **`VERACRYPT.EXE`** lautet. Abschließend finden wir ab Offset 76 den Little-Endian kodierten Prefetch Hash d6fd7e04. Dieser ist Teil des Dateinamens der betrachteten Prefetch-Datei.

File Information

An den Prefetch File Header schließt sich ab Offset 84 die Datenstruktur *Prefetch File Information* an. Diese enthält Informationen zum Aufbau dieser Prefetch-Datei sowie forensisch interessante Informationen zum Zeitpunkt der letzten Ausführung sowie zur Häufigkeit der Nutzung des Executables. Der File Information Block unterscheidet sich für die verschiedenen Windows-Varianten, Details zu den Datenfeldern für Windows Vista / Windows 7 finden Sie in Tabelle 4.6. Ab Windows 8.1 gibt es im Unterschied zu den Angaben in Tabelle 4.6 noch sieben weitere Zeitstempel zu den vorhergehenden Programmnutzungen. Das macht eine IT-forensische Zeitaussage noch einfacher.

Sektionen

Am Anfang des File Information Blocks gibt es insgesamt neun Datenfelder zur Lage und Größe der Sektionen A bis D:

- *Sektionen A und B*: beide Sektionen geben Informationen über die Ladezeit des Executables. Ab Offset $84 = 0x54$ bzw. $92 = 0x5C$ der Prefetch-Datei steht die Startadresse von Sektion A bzw. Sektion B, ab Offset 88 bzw. 96 die Anzahl der jeweiligen Einträge in Sektion A bzw. B.

Tabelle 4.6: File Information Datenfelder einer Prefetch-Datei in Windows Vista / Windows 7

Adresse (dez.)	Adresse (hex.)	Beschreibung
84-119	54-77	Insgesamt 9 Datenfelder mit je 32-Bit-Werten zu Offset und Größe der Sektionen A bis D
128-135	80-87	Zeitstempel der letzten Ausführung der Datei
152-155	98-9B	Anzahl der Aufrufe der Datei (Execution Counter, Run Count)

- *Sektion C*: diese Sektion enthält die Dateinamen der geladenen Dateien (also z.B. dynamische Bibliotheken). Die Dateinamen sind wie üblich als UTF-16 angegeben, und zwar als voller Windows-Geräte-Pfad (d.h. beginnend mit `\DEVICE\HARDDISKVOLUME[X]\...`). Ab Offset $100 = 0x64$ der Prefetch-Datei steht die Startadresse von Sektion C, ab Offset 104 die Länge von Sektion C in Bytes. Die Angaben in Sektion C sind forensisch interessant, weil sie die geladenen Bibliotheken und Hilfsdateien angeben.
- *Sektion D*: Ab Offset $108 = 0x6C$ der Prefetch-Datei steht die Startadresse von Sektion D, ab Offset $112 = 0x70$ die Anzahl der Einträge und ab Offset $116 = 0x74$ die Länge von Sektion D. Wir gehen gleich noch auf diese Sektion ein.

B

Beispiel 4.8: Prefetch File Information und Sektion C

Wir betrachten die gleiche Prefetch-Datei wie in Beispiel 4.7 und sehen uns die Datenfelder des File Information Blocks an.

```
$ xxd /mnt/Windows/Prefetch/VERACRYPT.EXE-047EFDD6.pf
[REMOVED]
0000050: 0000 0000 f000 0000 3800 0000 f007 0000  ........8.......
0000060: ad05 0000 0c4c 0000 961b 0000 a867 0000  .....L.......g..
0000070: 0100 0000 7a08 0000 0c00 0000 0100 0000  ....z...........
0000080: 9987 e34c f47a d101 0000 0000 0000 0000  ...L.z..........
0000090: 0000 0000 0000 0000 0200 0000 0100 0000  ................
[REMOVED]
0004c00: ffff ffff 8000 0000 0401 0201 5c00 4400  ............\.D.
0004c10: 4500 5600 4900 4300 4500 5c00 4800 4100  E.V.I.C.E.\.H.A.
0004c20: 5200 4400 4400 4900 5300 4b00 5600 4f00  R.D.D.I.S.K.V.O.
0004c30: 4c00 5500 4d00 4500 3200 5c00 5700 4900  L.U.M.E.2.\.W.I.
0004c40: 4e00 4400 4f00 5700 5300 5c00 5300 5900  N.D.O.W.S.\.S.Y.
0004c50: 5300 5400 4500 4d00 3300 3200 5c00 5200  S.T.E.M.3.2.\.R.
0004c60: 4900 4300 4800 4500 4400 3200 3000 2e00  I.C.H.E.D.2.0...
0004c70: 4400 4c00 4c00 0000 5c00 4400 4500 5600  D.L.L...\.D.E.V.
[REMOVED]
```

Ab Offset $0x54$ stehen die zwei Datenfelder für Sektion A (jedes Datenfeld hat die Länge 32-Bit): Sektion A beginnt ab Offset **0xf0** der Prefetch-Datei und enthält **0x38** Einträge. Ab Offset $0x5C$ stehen die zwei Datenfelder für Sektion B: diese beginnt ab Offset 0x07f0 der Prefetch-Datei und enthält 0x05ad Einträge. Es folgen ab Offset $0x64$ die zwei Datenfelder für Sektion C: diese beginnt ab Offset 0x4c0c der Prefetch-Datei und enthält 0x1b96 Einträge. Die erste in Sektion C dargestellte DLL ist `riched20.dll`, den vollen Windows-Gerätepfad sehen Sie ab Offset 0x4c0c.

Die Startadresse von Sektion D finden wir ab Offset $0x6C$ der Prefetch-Datei,

sie lautet 0x67a8. Offset 0*x*70 entnehmen wir, dass es nur einen Eintrag in Sektion D gibt.

Der Zeitstempel der letzten Programmausführung steht ab Offset 0*x*80, er lautet 0x01d17af44ce38799. Mittels DCode wandeln wir dies in das menschenlesbare Format um und erhalten 10. März 2016, 17:43:05 UTC. Das passt gut zu dem Zeitpunkt der letzten Änderung der Datei im Dateisystem, die gemäß Beispiel 4.6 10. März 2016, 17:44 Uhr UTC ist (am 10. März wird die Dateisystemzeit bezüglich UTC+1, also MEZ angezeigt). Die Anzahl der Programmstarts ist 2, wie wir dem Datenfeld ab Offset 0*x*98 entnehmen.

Sektion D

Sektion D gibt Informationen zu allen Partitionen, von denen Dateien verwendet werden. Daher heißt diese Sektion auch *Volume Information*. Die Datenfelder der Volume Information Sektion finden Sie in Tabelle 4.7 – bitte beachten Sie, dass die Offsets ab Beginn der Sektion D gezählt werden. Forensisch interessant ist, dass die Seriennummer der Partition samt Erstellungszeitpunkt sowie der Windows-Gerätepfad in dieser Sektion gespeichert werden.

Adresse (dez.)	Adresse (hex.)	Beschreibung
00-03	00-03	Offset zum Windows-Gerätepfad (ab Beginn von Sektion D)
08-15	08-0F	Zeitstempel der Erzeugung des Volumes
16-19	10-13	Seriennummer des Volumes
20-23	14-17	Offset zu Untersektion E (ab Beginn von Sektion D)
28-31	1C-1F	Offset zu Untersektion F (ab Beginn von Sektion D)

Tabelle 4.7: Volume Information Datenfelder einer Prefetch-Datei in Windows Vista / Windows 7

Untersektionen

Sektion D enthält zwei Untersektionen: in *Untersektion E* werden die NTFS-Dateireferenzen auf alle verwendeten Dateien (inkl. des aufgerufenen Executables) gespeichert. Damit gibt es selbst bei gelöschten Prefetch-Dateien wichtige Hinweise auf Informationen in der MFT über alle verwendeten Dateien. Untersektion E wird daher auch *NTFS File Reference* Sektion genannt. *Untersektion F* enthält Angaben zu allen zugehörigen Verzeichnissen – sie heißt auch *Directory Strings* Sektion.

B

Beispiel 4.9: Prefetch Volume Information

Wir betrachten die gleiche Prefetch-Datei wie in den Beispielen 4.7 sowie 4.8. Wir wissen bereits, dass Sektion D ab Offset 0*x*67*a*8 der Prefetch-Datei beginnt.

```
$ xxd /mnt/Windows/Prefetch/VERACRYPT.EXE-047EFDD6.pf
[REMOVED]
00067a0: 0000 d402 18ba d402 6800 0000 1700 0000  ........h.......
00067b0: d077 78dd 8769 d101 37e8 dda4 9800 0000  .wx..i..7.......
00067c0: 3802 0000 d002 0000 0c00 0000 3900 0000  8...........9...
[REMOVED]
0006810: 5c00 4400 4500 5600 4900 4300 4500 5c00  \.D.E.V.I.C.E./.
0006820: 4800 4100 5200 4400 4400 4900 5300 4b00  H.A.R.D.D.I.S.K.
0006830: 5600 4f00 4c00 5500 4d00 4500 3200 0000  V.O.L.U.M.E.2...
0006840: 0300 0000 4500 0000 f0bd d402 08be d402  ....E...........
[REMOVED]
0006890: d0a1 0000 0000 0800 8f4f 0000 0000 0100  .........O......
[REMOVED]
```

```
0006a70: b01f 0000 0000 0100 2500 5c00 4400 4500  ........%.\.D.E.
0006a80: 5600 4900 4300 4500 5c00 4800 4100 5200  V.I.C.E.\.H.A.R.
0006a90: 4400 4400 4900 5300 4b00 5600 4f00 4c00  D.D.I.S.K.V.O.L.
0006aa0: 5500 4d00 4500 3200 5c00 5000 5200 4f00  U.M.E.2.\.P.R.O.
0006ab0: 4700 5200 4100 4d00 2000 4600 4900 4c00  G.R.A.M. .F.I.L.
0006ac0: 4500 5300 0000 2f00 5c00 4400 4500 5600  E.S.../.\.D.E.V.
0006ad0: 4900 4300 4500 5c00 4800 4100 5200 4400  I.C.E.\.H.A.R.D.
[REMOVED]
```

Das Offset zum Windows-Gerätepfad finden wir ab Offset $0x0$ von Sektion D, also ab Offset $0x67a8$ der Prefetch-Datei. Den Wert 0x68 müssen wir zum Offset von Sektion D hinzuaddieren, um das Offset in der Prefetch-Datei zu erhalten. Es gilt 0x67a8 + 0x68 = 0x6810. Der Windows-Gerätepfad des Volumes lautet also `\DEVICE\HARDDISKVOLUME2`.

Den Erzeugungszeitpunkt des Volumes finden wir ab Offset $0x08$ von Sektion D, also ab Offset $0x67b0$ der Prefetch-Datei. Der 64-Bit-Wert lautet 0x01d16987dd7877d0, er entspricht dem Zeitpunkt 17. Februar 2016, 13:34:02 UTC. Die Seriennummer des Volumes finden wir ab Offset $0x10$ von Sektion D, also ab Offset $0x67b8$ der Prefetch-Datei. Sie lautet 0xa4dde837.

Das Offset zu Untersektion E finden wir ab Adresse $0x14$ in Sektion D, der Wert lautet **0x98**. Die NTFS-Dateireferenzen stehen daher ab Offset **0x67a8 + 0x98 = 0x6840** in der Prefetch-Datei. Nach einem 16 Byte langen Header finden wir die NTFS-Dateireferenzen aller geladenen DLLs, Hilfsdateien sowie des Executables selbst. Der Array ist ungeordnet. Mit Hilfe von `istat` aus dem Sleuthkit finden wir, dass die NTFS-Dateiadresse von `VeraCrypt.exe` ab Offset $0x6890$ zu finden ist. Der zugehörige MFT-Eintrag hat die Adresse 0xa1d0=41424 mit Sequenzummer 8.

Das Offset zu Untersektion F finden wir ab Adresse $0x1C$ in Sektion D, der Wert lautet 0x02d0. Die Verzeichnisnamen werden daher ab Offset 0x67a8 + 0x02d0 = 0x6a78 in der Prefetch-Datei abgelegt. Wir sehen, dass das erste genannte Verzeichnis der Programmordner \Program Files ist.

Tools

Kuhlee und Völzow [24] empfehlen den *Windows File Analyzer*[8] sowie den *Windows Prefetch Parser*[9] als Analysetools für Prefetch-Dateien. Der Windows File Analyzer ist nur für Windows erhältlich, er beherrscht auch die Analyse anderer interessanter Windows Artefakte (z.B. des Papierkorbs, siehe Abschnitt 4.5). Der Windows Prefetch Parser steht für Windows und Linux zur Verfügung. Er ist allerdings lt. Webseite nur für nicht-kommerzielle Zwecke in einer Demoversion kostenlos verfügbar (diese erwartet eine Authentifikationsdatei im Ordner des Windows Prefetch Parsers).

B

Beispiel 4.10: Prefetch Parser

In diesem Beispiel betrachten wir die Ausgabe des Prefetch Parsers. Wir wenden den Prefetch auf die in diesem Abschnitt betrachteten Windows-Systempartition an. Die Ausgabe besteht aus insgesamt 1017 Zeilen, so dass wir nur einen kleinen Auszug zeigen.

[8] http://www.forensikhacks.de/wfa
[9] http://www.forensikhacks.de/prefetch

```
========================
FIREFOX.EXE-7DFC8E4F.pf
========================

Executable Name: FIREFOX.EXE

Run count: 3
Last Executed: 2016-03-15 14:17:07.588132

Volume Information:
    Volume Name: \DEVICE\HARDDISKVOLUME2
    Creation Date: 2016-02-17 13:34:02.761620
    Serial Number: a4dde837

Directory Strings:
    \DEVICE\HARDDISKVOLUME2\USERS
[REMOVED]
    \DEVICE\HARDDISKVOLUME2\WINDOWS\WINSXS\X86_MICROSOFT...

Resources loaded:
1: \DEVICE\HARDDISKVOLUME2\USERS\EVE\DESKTOP\TOR BROWSER\BROWSER\
   MSVCR100.DLL
[REMOVED]
81: \DEVICE\HARDDISKVOLUME2\WINDOWS\WINDOWSSHELL.MANIFEST
```

Dieses Firefox Executable wurde nur drei Mal ausgeführt, zuletzt am 15.03.2016. Installiert wurde die Applikation vermutlich am 17.02.2016. Es lädt insgesamt 81 Ressourcen, wovon die erste Hinweise auf die Nutzung des Tor Browsers gibt.

Kontrollaufgabe 4.5: Prefetch Dateinamen

Erläutern Sie die Zusammensetzung des Dateinamens der Prefetch Dateien? Wieviele Dateien können pro Anwendung existieren?

K

Kontrollaufgabe 4.6: Prefetch Informationen

Welche forensischen Nutzen bietet die Analyse von Prefetch Dateien?

K

Kontrollaufgabe 4.7: Prefetch Aufbau

Erläutern Sie den Aufbau der Prefetch Dateien.

K

Kontrollaufgabe 4.8: Prefetch Sektionen

Welche Sektionen einer Prefetch-Datei enthalten relevante forensische Informationen?

K

4.5 Recycle.bin

Recycle.bin

Zum Abschluss des Studienbriefs betrachten wir im Zuge der Windows Artefakte den Recycle.Bin, genauer gesagt den Papierkorb des Windows Betriebssystems. Um dem Nutzer die Möglichkeit zu bieten, ein unbeabsichtigtes Löschen von Dateien rückgängig zu machen, werden vom Nutzer gelöschte Dateien nicht sofort im Dateisystem gelöscht. Vielmehr verschiebt das Windows Betriebssystem diese vom Nutzer gelöschten Dateien in den Papierkorb. Dort verbleiben diese Dateien, bis der Nutzer den Papierkorb leert.

Hinweis

Allerdings ist die Löschung mittels Leerens des Papierkorbs ebenfalls nicht endgültig, denn wie Sie bereits aus dem Grundlagenmodul *Einführung in die digitale Forensik* wissen, bleiben Nutz- und Metadaten von Dateien im Dateisystem erhalten, da die verwendeten Datenstrukturen lediglich für eine erneute Allokation freigegeben werden. Sie werden nicht sofort überschrieben. In der folgenden Diskussion gehen wir vom erstgenannten Fall aus, d.h. der Nutzer löscht mit Betriebssystemmitteln Dateien, so dass diese in den Papierkorb verschoben werden.

Prinzip

Löscht ein Nutzer eine Datei, wird die originale Datei in einen versteckten Systemordner verschoben und automatisch vom Betriebssystem umbenannt – dazu später mehr. Ab Windows Vista trägt der Systemordner den Namen `$Recycle.Bin`, in älteren Windows Versionen finden Sie diesen unter dem Namen `Recycler`. Inhalt des Papierkorbs sind nutzerspezifische Verzeichnisse, deren Namen die jeweilige SID des Nutzers ist. Daher kann eine Aussage getroffen werden, welcher Nutzer eine bestimmte Datei gelöscht hat. Betrachten wir zunächst Lage und Inhalt des Papierkorbs an Hand von Beispiel 4.11.

B

Beispiel 4.11: Inhalt des Systemordners $Recycle.Bin

Wir hängen eine Arbeitskopie eines sichergestellten Datenträgers mit Windows 7 in den Ordner `/mnt` unserer IT-forensischen Workstation ein. Wir navigieren in das Wurzelverzeichnis der Systempartition und lassen uns den Inhalt des Wurzelverzeichnisses anzeigen. Der Systemordner `$Recycle.Bin` ist farblich hervorgehoben.

```
$ cd /mnt
$ ls -l
[REMOVED]
drwxrwxrwx 1 root root          0 Feb 17  2016 Recovery
drwxrwxrwx 1 root root          0 Feb 17  2016 $Recycle.Bin
drwxrwxrwx 1 root root       4096 Mar 15  2016 System Volume Information
drwxrwxrwx 1 root root       4096 Feb 17  2016 Users
drwxrwxrwx 1 root root      16384 Mar 10  2016 Windows
```

Im nächsten Schritt wechseln wir in den Papierkorb und betrachten dessen Inhalt. Aktuell gibt es genau ein Unterverzeichnis, dessen Verzeichnisname die SID des Nutzers mit der relativen ID (RID) 1001 ist. Eine Verknüpfung zum Nutzernamen erreichen wir über die Windows Registry. Hinweis: Denken Sie daran, dass dem Dollar-Zeichen eine Escape-Sequenz (`\`) in der Kommandozeile vorangestellt werden muss, um in das Verzeichnis wechseln zu können.

```
$ cd \$Recycle.Bin
$ ls -l
drwxrwxrwx 1 root root 4096 Mar 15  2016 S-1-5-21-1650571544-2262814165-
                                          2515253795-1001
```

Wiederherstellung

Die Wiederherstellung einer mit Betriebssystemmitteln gelöschten Datei funktioniert wie folgt. Das Windows-Betriebssystem legt zusätzlich zur originalen gelöschten Datei eine Metadatei in dem nutzerspezifischen Verzeichnis im Papierkorb an. Die Metadatei enthält die notwendigen Informationen über die gelöschte Datei zu deren Wiederherstellung. Der Dateiname im Papierkorb der originalen Datei beginnt mit dem String `$R`, der Dateiname der zugehörigen Metadaten-Datei mit `$I`. Einige solcher Dateien sehen wir uns in Beispiel 4.12 an.

B

Beispiel 4.12: Dateien im nutzerspezifischen Papierkorb

Wir betrachten den Inhalt des Papierkorbs des Nutzers mit der relativen ID 1001.

```
$ ls -l /mnt/$Recycle.Bin/S-1-5-21-1650571544-2262814165-2515253795-1001
total 4929
-rwxrwxrwx 1 root root     129 Feb 17 14:42 desktop.ini
-rwxrwxrwx 1 root root     544 Mär 15 14:39 $I2CCU9N.lnk
-rwxrwxrwx 1 root root     544 Mär 15 14:40 $IK8KQW9.exe
-rwxrwxrwx 2 root root     544 Mär 15 14:40 $IP1RKA4.jpeg
-rwxrwxrwx 1 root root     544 Mär 10 15:46 $ITHR6AR.exe
-rwxrwxrwx 1 root root    1591 Mär 10 15:47 $R2CCU9N.lnk
-rwxrwxrwx 1 root root 1402992 Mär 10 18:39 $RK8KQW9.exe
-rwxrwxrwx 2 root root   29012 Mär 10 12:04 $RP1RKA4.jpeg
-rwxrwxrwx 1 root root 3598570 Mär 10 15:21 $RTHR6AR.exe
```

Wir sehen insgesamt 4 gelöschte Dateien: einen gelöschten Link, eine gelöschte Bilddatei sowie zwei gelöschte ausführbare Dateien. Die Metadaten-Datei hat jeweils eine Größe 544 Byte. Die beiden Dateien, die zu dem gelöschten Bild gehören, sind farblich hervorgehoben. Das gelöschte Bild ist mit ca. 29 KiB relativ klein.

Metadaten-Datei

Wie aus Beispiel 4.12 ersichtlich, hat eine Metadaten-Datei eine feste Größe von 544 Byte. Tabelle 4.8 gibt die Datenfelder einer Metadaten-Datei an. Da die gelöschte Datei vollständig vorliegt, bedeutet Wiederherstellung hier die Rückführung in das ursprüngliche Verzeichnis. Dazu wird der volle Verzeichnispfad inklusive des ursprünglichen Namens der gelöschten Datei benötigt. Der volle Verzeichnispfad steht wie für Windows üblich als UTF-16 kodierter String in der Metadaten-Datei (je nach Version ab Offset 24 oder Offset 28). Ebenso finden wir die Größe der originalen Datei (Offset 8) sowie den Zeitstempel der Dateilöschung als Windows-FILETIME (Offset 16). Ab Windows 10 wird die Version 2 des Datenformats genutzt, in diesem Fall befindet sich eine zusätzliche Angabe über die Länge des Dateinamens an Offset 24.

Adresse (dez.)	Adresse (hex.)	Beschreibung
00-07	00-07	Version (1 oder 2)
08-15	08-0F	Größe der originalen Datei
16-23	10-17	Zeitstempel der Löschung der originalen Datei
Version 1		
24-543	18-21F	Name inklusive Pfad der Originaldatei (UTF-16)
Version 2		
24-27	18-1B	Anzahl der Zeichen des Dateinamens
28-543	1C-21F	Name inklusive Pfad der Originaldatei (UTF-16)

Tabelle 4.8: Aufbau einer Metadaten-Datei `$I` des Windows Recycle.Bins [27].

B

Beispiel 4.13: Analyse einer Metadaten-Datei

Wir analysieren die Datenfelder der Metadaten-Datei $IP1RKA4.jpeg aus dem vorangegangenen Beispiel 4.12.

```
$ cd /mnt/$Recycle.Bin/S-1-5-21-1650571544-2262814165-2515253795-1001

$ xxd $IP1RKA4.jpeg | less
0000000: 0100 0000 0000 0000 5471 0000 0000 0000  ........Tq......
0000010: 704a 8d2d c07e d101 4300 3a00 5c00 5500  pJ.-.~..C.:.\.U.
0000020: 7300 6500 7200 7300 5c00 6500 7600 6500  s.e.r.s.\.e.v.e.
0000030: 5c00 4400 6f00 7700 6e00 6c00 6f00 6100  \.D.o.w.n.l.o.a.
0000040: 6400 7300 5c00 5300 6500 7200 7600 6500  d.s.\.S.e.r.v.e.
0000050: 7200 2d00 3100 6100 6500 3200 6300 3500  r.-.1.a.e.2.c.5.
0000060: 3500 6500 3200 6200 3600 6500 6300 3000  5.e.2.b.6.e.c.0.
0000070: 3000 6600 2e00 6a00 7000 6500 6700 0000  0.f...j.p.e.g...
0000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
REMOVED
00001e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000200: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000210: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

An Offset 0 bis 7 finden wir den Wert 0x01, es handelt sich daher um eine Metadaten-Datei der Version 1 (z.B. Windows 7 Betriebssystem). Die Größe der Originaldatei finden wir in Little-Endian kodiert an Offset 8-15, der Wert ist 0x7154=29012 Byte. Diese Angabe ist konsistent zur Dateigröße der Datei $RP1RKA4.jpeg aus Beispiel 4.12.

Nach dem Zeitstempel an Offset 16 bis 23 (704a 8d2d c07e d101) wurde die Datei am 15. März 2016, um 13:40 UTC gelöscht. Auch das passt zu dem Zeitstempel im Dateisystem der Metadaten-Datei aus Beispiel 4.12.

Den vollständigen Pfad der Datei sehen Sie ab Offset 24. Die originale Datei befand sich demnach im Verzeichnis C:\Users\eve\Downloads unter dem Namen Server-1ae2c55e2b6ec00f.jpeg.

Anti-Forensik

Abschließend zwei kurze Hinweise zu möglichen anti-forensischen Maßnahmen, denn der Nutzer kann das Anlegen von Dateien im `$Recycle.Bin`-Ordner auch mit Mitteln des Betriebssystems verhindern:

1. Löscht der Nutzer eine Datei mit gedrückter SHIFT-Taste, wird diese nicht in den Papierkorb verschoben, sondern sofort gelöscht.
2. Eine systemweite Einstellung zum sofortigen Löschen bietet die *Nuke on Delete* Einstellung innerhalb des Software Hives der Registry. Dazu aktiviert man den entsprechenden Registry-Schlüssel[10].

[10] Software\Microsoft\Windows\CurrentVersion\Explorer\BitBucket\NukeOnDelete

In diesen Fällen bleibt nur die Analyse mit Hilfe der Informationen bzw. Metadaten auf Dateisystemebene.

Kontrollaufgabe 4.9

Welche Informationen über die Datei können aus dem Recycle.Bin gewonnen werden?

K

4.6 Schattenkopien

Im letzten Abschnitt des Studienbriefs betrachten wir die Funktion von sogenannten Schattenkopien (*Volume Shadow Copies*). Vor allem gehen wir natürlich der Frage nach, welchen Mehrwert die Analyse von Schattenkopien im Rahmen einer forensischen liefern können.

Volume Shadow Copies

Schattenkopien sind ältere Versionen einer Datei, die zur Laufzeit im Hintergrund vom Windows Betriebssystem angelegt werden. Genauer gesagt handelt es sich dabei um Wiederherstellungspunkte von Dateien bzw Verzeichnissen. Ursprünglich wurde diese Funktion für Systemdateien eingeführt, um das Betriebssystem wieder in einen konsistenten Zustand versetzen zu können. Beispielsweise durch falsche Konfiguration oder Treiberproblemen. Seit Windows Vista wurde diese Beschränkung aufgehoben, so dass es möglich ist für beliebige Dateien Schattenkopien zu erstellen. Eine weitere Neuerung der Funktion war, dass nicht mehr der komplette Inhalt der Datei gesichert wurde, sondern lediglich das Delta der letzten Änderung. In forensischer Hinsicht ist die Funktion aus folgendem Grund interessant: Mit Hilfe der Auswertung von Volumenschattenkopien kann auf ältere Kopien von Dateien zugegriffen werden. Wird die originale Datei geändert oder gar gelöscht, kann diese wiederhergestellt werden. Dies gilt auch für *sicher* gelöschte Dateien mittels spezieller Software. Beachten Sie dabei, dass die Sicherungspunkte automatisch durch das System erstellt werden und nicht manuell.

sicher gelöschte Daten

Die Schattenkopien werden im Verzeichnis *System Volume Information* gespeichert und genau auf der Partition, auf der sich die originalen Dateien befinden. Das bedeutet also: existieren mehrere Partitionen, so existiert auf jeder Partion ein seperates Verzeichnis mit Schattenkopien.

Zu beachten ist, dass die Nutzung dieser Funktion durch den Anwender des Systems konfiguriert werden kann. Ob Volumenschattenkopien existieren lässt sich anhand von Systemeinstellungen prüfen. Windows nutzt dazu den Dienst *Volumenschattenkopie*. Des Weiteren lässt Einstellungen zum Dienst in den erweiterten Systemeigenschaften konfigurieren. Hierbei kann die Funktion für einzelne Partitionen aktiviert bzw. deaktiviert werden.

Einen komfortablere Möglichkeit die Partionen auf mögliche Schattenkopien zu untersuchen bietet die kostenlose Software *Shadow Explorer*[11], womit sich ältere Versionen einzelner Ordner und Dateien problemlos extrahieren lassen (vgl. Abbildung 4.6).

4.7 Zusammenfassung

In diesem Studienbrief haben Sie mehrere ausgewählte Artefakte innerhalb des Windows Betriebssystem kennen gelernt, die im Zuge einer forensischen Untersuchen einen Mehrwert zur Klärung der vorliegenden Zweifelsfrage beitragen

[11] http://www.shadowexplorer.com/downloads.html

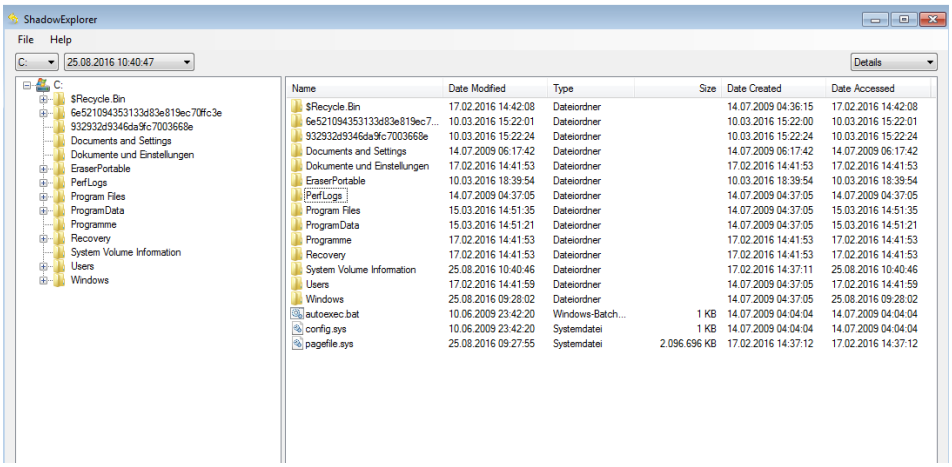


Abb. 4.7: Shadow Explorer.

können. Wie Sie jedoch in der in Abschnitt 4.2 gezeigten Übersichtsgrafik (Abbildung 4.1) erkennen, sind wir bewusst nicht auf alle dargestellten Artefakte eingegangen. Dies hat mehrere Gründe. Zum Einen ist es, durch die ständige Weiterentwicklung des Windows Betriebssystems, möglich mehrere Studienbriefe mit diesem Themenbereich zu füllen. Zum Anderen sollen Sie sich nach dem Durcharbeiten dieses Studienbriefs um die Existenz solcher Artefakte bewusst werden und sich einen Überblick darüber verschafffen, welche Informationen Sie aus diesen gewinnen können. Zwar erfordern mögliche anti-forensische Maßnahmen zur Vermeidung solcher Artefakte einen technisch versierten Nutzer und die Manipulation dieser Art von Spuren erfordert einen nahezu unrealistisch hohen Aufwand. Nichtsdestotrotz müssen Sie diese Maßnahmen, sei es über Einstellungsoptionen in der Registry oder durch manuelles Entfernen der Artefakte, im Auge behalten.

Verzeichnisse

I. Abbildungen

II. Beispiele

III. Definitionen

IV. Kontrollaufgaben

V. Tabellen

VI. Literatur

[1] Oleg Afonin and Yuri Gubanov. Catching the ghost: how to discover ephemeral evidence with live ram analysis. Technical report, Belkasoft Research, 2013.

[2] Ronny Bodach. Sqlite 3 datenbanken - wiederherstellung gelöschter records. Technical report, http://www.tatortgruppe.de/documents/EWF%20-%20Recovering%20SQlite.pdf, zuletzt aufgerufen am: 03.08.2012.

[3] D Brezinski and Tom Killalea. Guidelines for evidence collection and archiving. Technical report, 2002.

[4] Harlan Carvey. *Windows Registry Forensics - Advanced digital forensic analysis of the Windows Registry*. Elsevier, 2011.

[5] Michael Cohen, Simson Garfinkel, and Bradley Schatz. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. *digital investigation*, 6:S57–S68, 2009.

[6] Kevin Conlan, Ibrahim Baggili, and Frank Breitinger. Anti-forensics: Furthering digital forensic science through a new extended, granular taxonomy. *Digital Investigation*, 18, 2016.

[7] Andreas Dewald and Felix C. Freiling (Hrsg.). *Forensische Informatik*. Books on Demand Verlag, 2015.

[8] Andreas Dewald, Felix Freiling, and Tim Weber. Design and Implementation of a Forensic Documentation Tool for Interactive Command-line Sessions. In *Proceedings of the 7th International Conference on IT Security Incident Management and IT Forensics (IMF)*, pages 62–80, 2011.

[9] Sanderson Forensics. Recovering deleted records from an SQLite database. https://sandersonforensics.com/forum/content.php?222-Recovering-deleted-records-from-an-SQLite-database, 2015. aufgerufen am 18.11.2017.

[10] Forensicswiki. Windows Prefetch File Format. http://www.forensicswiki.org/wiki/Windows_Prefetch_File_Format. Zugriff am 02.08.2016.

[11] J.C. Foster and V. Liu. Catch me if you can... In *Blackhat Briefings 2005*, Las Vegas, NV, August 2005. URL http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-foster-liu-update.pdf.

[12] Mozilla Foundation. MDM Webdocs. https://developer.mozilla.org/en-US/, 2017. aufgerufen am 20.11.2017.

[13] Mozilla Foundation. Profiles - Where Firefox stores your bookmarks, passwords and other user data. https://support.mozilla.org/en-US/kb/profiles-where-firefox-stores-user-data, 2017. aufgerufen am 21.11.2017.

[14] Institut für Sachverständigenwesen. Empfehlungen zum Aufbau eines Sachverständigengutachtens. https://www.ifsforum.de/fileadmin/user_upload/Merkblatt_zum_Aufbau_eines_Sachverstaendigengutachtens_08-2014.pdf, 2014. Zugriff am 08.08.2016.

[15] Bundesamt für Sicherheit in der Informationstechnik. Leitfaden "IT-Forensik". Technical report, http://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Internetsicherheit/Leitfaden_IT-Forensik_pdf.pdf?__blob=publicationFile, zuletzt aufgerufen am: 18.06.2014.

[16] A. Geschonneck. *Computer-Forensik (iX Edition): Computerstraftaten erkennen, ermitteln, aufklären*. dpunkt.verlag, 2014. ISBN 9783864914904. URL https://books.google.de/books?id=s89NDAAAQBAJ.

[17] Alexander Geschonneck. *Computer Forensik*. dpunkt Verlag, 2014.

[18] GitHub. Windows Prefetch File (PF) format. https://github.com/libyal/libscca/blob/master/documentation/Windows%20Prefetch%20File%20(PF)%20format.asciidoc. Zugriff am 02.08.2016.

[19] Grugq. The art of defiling: defeating forensic analysis. In *Blackhat briefings 2005*, Las Vegas, NV, August 2005. URL http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-grugq.pdf.

[20] Ryan Harris. Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem. *Digital Investigation*, 3, 2006. ISSN 1742-2876. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

[21] N.A. Hassan and R. Hijazi. *Data Hiding Techniques in Windows OS: A Practical Approach to Investigation and Defense*. Elsevier Science, 2016. ISBN 9780128044964. URL https://books.google.de/books?id=sy2lCgAAQBAJ.

[22] Keith J. Jones, Richard Bejtlich, and Curtis W. Rose. *Real Digital Forensics: Computer Security and Incident Response*. Addison-Wesley Professional, 2006.

[23] L. Kuhlee and V. Völzow. *Computer-Forensik Hacks : (Buch mit E-Book)*. Hacks series. O'Reilly, 2012. ISBN 9783868991215. URL https://books.google.de/books?id=bQblShv0v2EC.

[24] Victor Kuhlee, Lorenz & Völzow. *Computer Forensik Hacks*. O'Reilly, 2012.

[25] Michael Hale Ligh, Andrew Case, Jamie Levy, and Aaron Walters. *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. Wiley Publishing, 1. edition, 2014.

[26] Joachim Metz. Windows explorer thumbnail cache database format specification. https://github.com/libyal/libwtcdb/blob/master/documentation/Windows%20Explorer%20Thumbnail%20Cache%20database%20format.asciidoc, 2015. aufgerufen am 12.10.2017.

[27] Joachim Metz. The windows recycle.bin file format. https://github.com/libyal/assorted/blob/master/documentation/Windows%20Recycle.Bin%20file%20format.asciidoc, 2016. aufgerufen am 02.11.2017.

[28] Microsoft. Windows registry information for advanced users. https://support.microsoft.com/en-us/kb/256986, 2011. aufgerufen am 02.07.2016.

[29] Microsoft. Inside the Registry. http://technet.microsoft.com/de-de/library/cc750583.aspx, letzter Zugriff: 19.09.2016, 2016.

[30] MozillaZine. Profile folder - Firefox. http://kb.mozillazine.org/Profile_folder_-_Firefox, 2015. aufgerufen am 21.11.2017.

[31] Christian SJ Peron and Michael Legary. Digital anti-forensics: emerging trends in data transformation techniques. In *Proceedings of*, 2005.

[32] M Rogers. Anti-forensics: the coming wave in digital forensics. *Retrieved September*, 7:2008, 2006.

[33] B Shirani. Anti-forensics. *High Technology Crime Investigation Association, http://www.aversion.net/presentations/HTCIA-02/anti-forensics.ppt*, 2002.

[34] SQLite.org. The sqlite database file format. Technical report, http://www.sqlite.org/fileformat.html, zuletzt besucht am: 17.11.2017.

[35] Stefan VöMel and Felix C. Freiling. A survey of main memory acquisition and analysis techniques for the windows operating system. *Digit. Investig.*, 8(1):3–22, July 2011. ISSN 1742-2876. doi: 10.1016/j.diin.2011.06.002. URL http://dx.doi.org/10.1016/j.diin.2011.06.002.

[36] Stefan Vömel and Johannes Stüttgen. An Evaluation Platform for Forensic Memory Acquisition Software. In Elsevier B.V., editor, *Proceedings of the 13th Annual DFRWS Conference*, pages 1–12, 2013.