

Entwicklungen in den Informations- und Kommunikationstechnologien

Herausgeber:
Friedrich-L. Holl

Band 4

IT-Forensik Ausgewählte Aspekte zu Sicheren Rechnerarchitekturen, FAT und NTFS

Autoren:

Klaus Hildebrandt
Stefan Humml
Igor Podebrad

Herausgeber: Prof. Dr. Friedrich-L. Holl, Fachhochschule Brandenburg

© 2011 Eigenverlag, Berlin
Redaktion: Friedrich-L. Holl
Satz: Martin Schüngel, Köln
Druck: digital business and printing gmbh, Berlin

ISSN 1863-5016

Alle Rechte vorbehalten, insbesondere die des öffentlichen Vortrags, der Rundfunk-
sendung und der Fernsehausstrahlung, der Übersetzung, der fotomechanischen
Wiedergabe, auch einzelner Teile, mit Ausnahme der in den §§ 53, 54 URG genannten
Sonderfälle.

Inhalt

| | |
|---|-----------|
| Igor Podebrad und Klaus Hildebrandt | |
| IT-forensische Designaspekte einer sicheren Rechnerarchitektur | 5 |
| Stefan Humml | |
| Das FAT Dateisystem | 19 |
| Stefan Humml | |
| Das NTFS Dateisystem | 43 |

IT-forensische Designaspekte

einer sicheren

Rechnerarchitektur

von **Igor Podebrad** und **Klaus Hildebrandt**

Abstract: Sicherheit theoretisch zu definieren und praktisch in das Design einer Rechnerarchitektur einfließen zu lassen, sind bereits zwei unterschiedlichen Dinge. Die Gewissheit, dass diese Überlegungen und umgesetzten Maßnahmen auch tatsächlich Wirkung zeigen, muss durch entsprechende Überprüfungen belegt werden. Hierzu müssen funktionale Mechanismen bereits bei der Entwicklung und Implementierung eines IT-Systems berücksichtigt werden. Diese Arbeit untersucht, welche Anforderungen aus IT-forensischer Sicht bei der Entwicklung einer sicheren Rechnerarchitektur zu stellen und wie diese praktisch zu bewerten sind. Mit der Vorstellung des Konstrukts der Haltbarkeitsspanne von IT-forensisch relevanten Informationen wird ein konkreter Lösungsansatz sowie ein flankierendes, methodisches Rahmenkonzept vorgestellt.

Inhaltsverzeichnis

| | |
|--|----|
| 1 Einleitung | 9 |
| 2 Anforderungen der IT-Forensik | 9 |
| 3 Hinweise auf IT-forensische Designaspekte | 11 |
| 3.1 Inhalte der Aufzeichnung: | |
| Zu protokollierende Aspekte in einer Rechnerarchitektur | 11 |
| 3.2 Speicherung der aufgezeichneten Informationen: | |
| Wie viel, wie lange und wo? | 13 |
| 3.3 Zugriff auf aufgezeichnete Inhalte: Wer darf was? | 14 |
| 3.4 Auswertung der aufgezeichneten Inhalte: Was soll wann verfügbar sein? .. | 15 |
| 3.5 Ableitungen von Maßnahmen: Was soll wann passieren? | 16 |
| 3.6 Referenzzeit: Wie und wo wird die Referenzzeit festgelegt? | 16 |
| 4 Schlussfolgerung | 17 |
| 5 Quellen | 17 |

1 Einleitung

Die Implementierung von Sicherheit in einem IT-System orientiert sich grundsätzlich an den zuvor definierten Sicherheitszielen und hat bei deren Erfüllung mehrere technische Ausprägungen zu berücksichtigen [PHK09]. Alle diese Ausprägungen zielen darauf ab, Sicherheit in einem IT-System bereits durch architekturelle Veränderungen an neuralgischen Punkten im Rechnersystem grundlegend zu etablieren, um somit bestimmte Formen von Angriffsvektoren im Vorfeld auszuschließen zu können. Diese Maßnahmen können als aktiver Beitrag zu der genannten Zielerreichung eingestuft werden, da ihre Existenz zur Laufzeit z.B. bestimmte Entitäten authentisiert oder im Rahmen einer Anti-X-Lösung bestimmten malignen Veränderungen [Ruto6] direkt vorbeugt. Dies bedeutet konkret, dass solche Schutzmaßnahmen im Sinne eines präventiven Ansatzes entwickelt werden und dann im Rahmen der Implementierung wirkungsvoll greifen.

IT-forensische Untersuchungen sind demgegenüber überwiegend ex-post Analysen. Um eine solche Aufklärung fragwürdiger Zustände in einem IT-System im Nachhinein überhaupt sinnvoll durchführen zu können, muss das Sicherheitsziel der Nachvollziehbarkeit umfänglich erfüllt werden, d.h. es müssen Informationen über die relevanten Vorgänge und Veränderungen im IT-System in angemessener Detailtiefe, in definiertem Umfang sowie dem entsprechenden Kontext verfügbar sein.

2 Anforderungen der IT-Forensik

Aus dieser scheinbar einfachen Anforderung an die Nachvollziehbarkeit von Ereignissen ergeben sich in der Praxis einige weitreichende Herausforderungen beim Design einer sicheren Rechnerarchitektur. Im Einzelnen betrifft dies folgende Aspekte:

Inhalte der Aufzeichnung.

Der Erfolg einer forensischen Untersuchung hängt sehr stark von den vorher gesammelten Informationen ab. Dies bedeutet, dass die Entscheidung über die Wichtigkeit der Information spekulativ erfolgen muss. Die Möglichkeit, einfach alles zu speichern, widerspricht meist den technischen Möglichkeiten bzw. der Wirtschaftlichkeit (siehe Abschnitt 3.2). Weiterhin muss die Aussagekraft der erhobenen Informationen berücksichtigt werden. Es ist leicht nachvollziehbar, dass manche Informationen isoliert betrachtet zunächst wertlos sind, jedoch in Kombinationen mit anderen Daten mitunter erhebliche Aussagekraft gewinnen können.

Speicherung der aufgezeichneten Informationen.

Wie bereits erwähnt, ist die Relevanz der Information von äußerster Wichtigkeit. Die Relevanz ist aber nicht konstant, d.h. sie nimmt mit der Zeit ab. Daraus folgt, dass die Entscheidung, wie lange bestimmte Informationen aufbewahrt werden müssen, nicht trivial ist. Ein weiterer Entscheidungsaspekt stellt der Vergleich bezüglich der Vor- und Nachteile einer zentralen bzw. der dezentralen Speicherung dar.

- Dauer der Speicherung: Wie lange und in welchem Umfang sollen bzw. müssen aufgezeichnete Informationen aufbewahrt werden?
- Ort der Speicherung: Sollen die relevanten Informationen in eine zentrale Daten-senke münden oder eher dezentral vorgehalten werden?

Zugriff auf aufgezeichnete Inhalte.

Die aufgezeichneten Inhalte stellen sensible Informationen dar. Umso wichtiger ist es, dass bei dem Design einer Sicherheitslösung detailliert dafür Sorge getragen wird, dass nur bestimmte funktionale Rollen Zugriff auf diese Daten haben.

Ableitungen von Maßnahmen.

Wie bereits erwähnt werden forensische Untersuchungen meist ex-post durchgeführt. Die Zeitspanne zwischen einem Vorfall und der Reaktion (z.B. der Untersuchung oder der Einleitung von Gegenmaßnahmen) bestimmt weitgehend die Verluste, welche durch einen Vorfall verursacht werden. Es erscheint also sinnvoll und wünschenswert, einige Maßnahmen on-the-fly oder zumindest unmittelbar nach der Feststellung bestimmter Ergebnisse auf Basis der eigenen Auswertungen einzuleiten. Somit stellt sich nun die Frage, wie bestimmte Ereignisse zu bewerten sind und welche Maßnahmen aus ihnen direkt resultieren.

Referenzzeit.

Die zu entwickelnde Sicherheitsarchitektur muss zwingend das Problem der Referenzzeit lösen. Konkret bedeutet dies, dass es eine Zeitreferenz geben muss, die unabhängig von der Zeit im kompromittierten System ist. Hintergrund hierfür ist die Notwendigkeit, bei einer IT-forensischen Untersuchung eine nicht manipulierbare Zeit kontinuierlich verfügbar zu haben.

Auswirkungen auf die Performanz des IT-Systems.

Die Aufzeichnung von Ereignissen bindet Systemressourcen, die zunächst nicht für andere Vorgänge zur Verfügung stehen. Weiterhin kann der Zustand eintreten, dass bestimmte Vorgänge langsamer ablaufen als im Zustand ohne entsprechende Aufzeichnung.

Alle diese Fragen scheinen ein auf den ersten Blick kaum lösbares Dilemma zu erzeugen. Dies wiegt umso schwerer, da man vor dem Hintergrund, Sicherheit als integralen Bestandteil einer Rechnerarchitektur implementieren zu wollen, sehr klaren Restriktionen unterworfen ist. Diese sind unter anderem

- die Kompromittierbarkeit der gewählten Sicherheitsarchitektur: Es muss sichergestellt, dass die gewählte Lösung nicht selbst kompromittiert werden kann. Eine Kompromittierung kann z.B. durch die Benutzung der Komponenten des überwachten Rechners entstehen.
- die Wechselwirkungen durch die Einführung der Sicherheitsarchitektur: Ergeben sich Wechselwirkungen mit der eigentlichen Rechnerarchitektur bzw. ist die Einführung der Sicherheitsarchitektur mit funktionalen Einschränkungen verbun-

den oder widerspricht die vorgeschlagene Lösung gegebenenfalls sogar dem zugrundeliegenden Rechnermodell?

- die zeitliche Akzeptanz: Im Idealfall soll die Datensammlung und Auswertung die Funktionstüchtigkeit des Produktivsystems nicht beeinträchtigen. Im Realfall ist vermutlich mit zusätzlichen Belastungen durch die Datensammlung und der on-the-fly-Auswertung zu rechnen. Man sollte auf jeden Fall dafür sorgen, dass die definierte Verfügbarkeit erhalten bleibt.

3 Hinweise auf IT-forensische Designaspekte

Bevor konkrete technische Lösungsansätze diskutiert werden können, muss zunächst eine hinreichend umfangreiche Liste der relevanten aufzuzeichnenden Inhalte definiert werden. Hierbei sollte beachtet werden, dass es im Gegensatz zur klassischen Forensik, d.h. der nachgelagerten Untersuchung von Vorfällen, im Rahmen einer sicheren Rechnerarchitektur notwendig ist, bestimmte Ereignisse bereits zur Laufzeit analysieren zu können. Diesem Aspekt kommt besondere Bedeutung zu, da im Falle einer erfolgreichen Lösung kritische Folgeprobleme wie z.B. das Mengenproblem bei der Aufzeichnung und Auswertung von Informationen bereits vorab minimiert werden können.

3.1 Inhalte der Aufzeichnung:

Zu protokollierende Aspekte in einer Rechnerarchitektur

Sremack hat vorgeschlagen, im Rahmen der forensischen Untersuchung von real-time Systemen bei der Sammlung von relevanten Informationen zu allererst die Priorität auf nicht volatile Informationen zu legen [Sreo5]. Aus seiner Sicht sind dabei

- der Inhalt von Registern und Zwischenspeichern (cache memory),
- der Inhalt von RAM und ROM,
- der Zustand von systemischen Zeitservices wie z.B. der Erkennung von time delays und time-outs,
- der Zustand der geladenen Module,
- der Zustand der Netzwerkverbindungen,
- der Zustand laufender Prozesse und threads sowie
- der Inhalt und Zustand der Speichermedien

für die weiteren Analysen erfolgskritisch.

Das primär lohnende Gebiet bezüglich der Gewinnung wichtiger Informationen in IT-forensischer Hinsicht ist in der Tat der Arbeitsspeicher. Seine grundlegende Bedeutung erklärt sich damit, dass bei den aktuell verbreiteten Rechnerarchitekturen ausnahmslos alle Vorgänge im Rechner über den Arbeitsspeicher laufen, unabhängig davon ob der Prozessor involviert ist wie z.B. bei der Ausführung von Prozessen oder

auch nicht wie z.B. bei direct memory access (DMA) von unterschiedlichen Sekundärkomponenten. Somit erscheint es sinnvoll, genau an dieser Stelle im System die Möglichkeit einer Aufzeichnung einzuführen. Abhängig von der gewählten Speicherarchitektur muss nun wiederum zwischen den dort befindlichen Daten und Instruktionen unterschieden werden. Für eine IT-forensische Untersuchung ist es z. B. sehr wohl interessant zu wissen, ob und wann der Programmcode einer Anwendung verändert wurde. Rutkowska hat in ihren Arbeiten zu BluePill gezeigt, dass dies ein durchaus realistisches Angriffsszenario darstellt [Rut]. Möchte man jedoch im Rahmen der Sicherheitsarchitektur eine Veränderung der Befehle eines Programms nicht erst im Nachhinein bemerken, sondern diesen Zustand frühestmöglich erkennen und sofort daraus bestimmte Handlungen ableiten können, so müssen sowohl der Befehl an sich wie auch die möglichen Wege, diesen manipulieren zu können beobachtet werden. Es ist in diesem Zusammenhang auch unerheblich, ob die betrachtete Speicherarchitektur ein von-Neumann-Typ oder ein Harvard-Typ ist, da beide Typen keine Einschränkungen hinsichtlich Lese-/Schreibzugriff kennen. Die von Sremack vorgeschlagene prioritäre Aufzeichnung von nicht-volatilen Informationen kann daher zum überwiegenden Teil mit einer Aufzeichnung des Arbeitsspeichers erfasst werden [Sre05]. Problem hierbei ist jedoch, dass die Kenntnis davon alleine noch keine hinreichende Transparenz über den Zustand der Programm-Befehle erzeugt!

Bei den zu verarbeitenden Daten im Arbeitsspeicher stellt sich nämlich prinzipiell die Frage, welche Daten ‚gutartig‘ und welche ‚böartig‘ sind, da diese Daten gemeinsam mit den entsprechenden Befehlen tatsächlich erst ungewollte Veränderungen verursachen können. An dieser Stelle wird sehr schnell deutlich, dass eine isolierte Aufzeichnung von Veränderungen im Datenspeicher ebenfalls noch keinen unmittelbaren Erkenntnisgewinn bringt. Vielmehr ergibt sich die Notwendigkeit, Veränderungen an Daten im erweiterten Kontext zu bewerten, um die entsprechenden Aktionen dann eventuell nach Gut- bzw. Böartigkeit unterscheiden zu können. Wie verhält es sich mit den anderen Komponenten einer klassischen Rechnerarchitektur, wenn es um die Frage geht, ob dort sinnvollerweise Informationen in Hinblick auf eine IT-forensische Analyse erhoben werden sollen? Für die im System befindlichen Prozessoren ist die Antwort recht einfach: Da bei den Prozessoren deren Funktionalität im Sinne von elektrischen Schaltungen fest in Silikon gegossen ist, kann sich diese zunächst nicht verändern. Die einzige Möglichkeit hierzu ergibt sich, wenn die Befehlssequenzen der Anwendung verändert werden, was jedoch bei aktuell verwendeten Rechnerarchitekturen nur im Arbeitsspeicher des Systems passieren kann!

Bei den Verbindungen zwischen den Prozessoren und den Speicherkomponenten wäre eine Aufzeichnung theoretisch und praktisch möglich. Allerdings muss auch hier bedacht werden, dass alle potentiellen Aktionen im System, unabhängig von ihren Auswirkungen, zur Realisierung den Arbeitsspeicher (sogar meist in Verbindung mit dem Prozessor) benötigen.

3.2 Speicherung der aufgezeichneten Informationen: Wie viel, wie lange und wo?

Das zentrale Problem bei der Speicherung von aufgezeichneten Informationen besteht darin, den Zielkonflikt zwischen unbegrenzter quantitativer sowie zeitlicher Vorhaltung und dem damit verbundenen, enorm anwachsenden Speicherbedarf aufzulösen. Aus IT-forensischer Perspektive ist es per se wünschenswert, Informationen so lange wie möglich zu bevorraten, da diese Daten für die Überprüfung der Bewertung von Angriffsvektoren zur Verfügung stehen und Klarheit darüber geben können, ob diese in der Vergangenheit auf dem System wirksam waren. Die Umsetzung dieser Idee stößt in der Praxis jedoch sehr schnell an technische Grenzen, da auf keinem System Sekundärspeicher unbegrenzt verfügbar ist. Ansätze wie Komprimierung oder Auslagerung lösen das Problem nur scheinbar, denn es generiert sich im Laufe der Zeit eine schier unendliche Menge an Informationen, die auf ihre Auswertung warten.

Theoretisch wäre das Mengen-Problem mit einem System lösbar, das das zu überwachende System leistungsmäßig bei weitem übertrifft. Ein solches System wäre dann in der Lage alle Daten just-in-time auszuwerten. Die Realisierung dürfte allerdings an der Größe scheitern.

Idealerweise kann das Problem zumindest dadurch reduziert werden, indem man sich überlegt, welche Informationen welche zeitliche Relevanz im Sinne einer ‚Haltbarkeitsspanne‘ haben und welche tatsächlich für eine Bewertung von sicherheitskritischen Ereignissen benötigt werden.

Ein Beispiel (siehe Abbildung 1) soll dies verdeutlichen: Zum Zeitpunkt t-6 wurde im Befehlsfolge der Befehl x durch ein Schadprogramm von seinem originären Zustand 0 in den Zustand 1 verändert. Vier Zeitperioden später, d.h. in t-2 wurde diese Änderung von dem Schadprogramm zwecks Verwischung der Spuren wieder rückgängig gemacht.

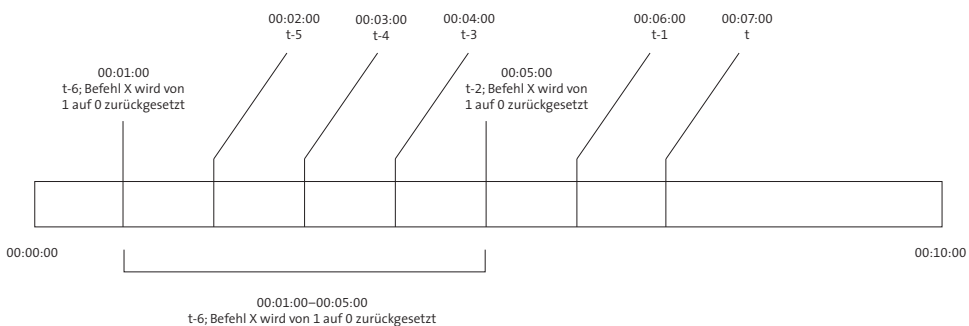


ABBILDUNG 1: ZEITLICHER ABLAUF EINES BEISPIELANGRIFFS

Ansonsten passieren keine weiteren sicherheitsrelevanten Aktivitäten auf dem gesamten System. Folglich ist also die »Haltbarkeitsspanne« dieser Information genau vier Zeitperioden lang, für alle anderen Zeitpunkte hat diese Information keine Bedeutung, insofern dies die einzige Änderung an der Befehlsfolge darstellt (was für den Moment in unserem Beispiel als solches so angenommen wird). Vor diesem Hintergrund könnte man sich nun entscheiden, dass lediglich die Zeitspanne von t-6 bis t-2 tatsächlich im Sekundärspeicher verbleibt und alle anderen Perioden wegen fehlender Relevanz verworfen werden. Dieses Entscheidungsprinzip hat zwei wesentliche Vorteile – die Verminderung des Bedarfs an Sekundärspeicher und die Vorselektion von möglichen sicherheitskritischen Ereignissen für die Phase der Auswertung. Dies wird jedoch mit dem Nachteil erkaufte, dass Informationen, die aus heutiger Sicht ohne Relevanz sind, in Zukunft jedoch aufgrund weiterer neuer Informationen plötzlich eine Bedeutung erfahren, dann unwiderruflich verschwunden sind! Dieser Nachteil ist allerdings bei genauerer Betrachtung nicht ganz so gravierend, wenn man sich die Konsequenzen überlegt. Die Information, dass die Befehlssequenz verändert wurde¹, ist in erster Linie für die Ableitung von Maßnahmen wichtig. Wenn keine Maßnahmen ergriffen werden, dann muss diese Information streng genommen auch gar nicht aufbewahrt werden. Resultiert aus einer solchen Information eine Aktivität, dann wird man sinnvollerweise die Information zwar aufbewahren, diese jedoch verdichten, d.h. nicht die gesamten Rohdaten speichern, sondern eine bewertete Information über den beobachteten Vorgang ablegen (wie z.B. Veränderung der Befehlssequenz² in t-6 bis t-2, Befehl x). Als wesentliches Zwischenergebnis kann somit festgehalten werden, dass die Grundidee der Aufbewahrung von selektierten und bewerteten Ereignissen signifikante Vorteile gegenüber den herkömmlichen Ansätzen aufweist.

Als Ort der Speicherung empfiehlt sich eine Lokation, auf die aus dem kompromittierten System heraus nicht direkt zugegriffen werden kann. Das ursächliche Problem besteht nämlich darin, dass ein Angreifer unmittelbar versuchen wird, die Spuren seines Tuns zu eliminieren, indem er die Logs entsprechend löscht oder manipuliert. Schutzmechanismen, die sich z.B. auf den Schutz der Log-Files mittels ACLs verlassen, sind per se unwirksam, da der Angreifer in der Regel im Rahmen des erfolgreichen Angriffs administrative Rechte auf dem System erlangt hat und somit diese Barrieren mühelos überwindet. Dies kann verhindert werden, indem beispielsweise die Aufzeichnung auf einem isolierten bzw. externen System stattfindet. Ein ähnliches Konzept ist in Petroni et al. zu finden [PFMA09].

3.3 Zugriff auf aufgezeichnete Inhalte: Wer darf was?

Unter der Annahme, dass im betrachteten System alle sicherheitsrelevanten Informationen optimiert und nicht manipulierbar aufgezeichnet werden, stellt sich nun

1 an der Stelle ist irrelevant, ob sich die Reihenfolge der Befehle oder die Befehle selbst verändert wurden

2 entspricht in etwa der Programmänderung

die Frage, welche Entität dieses Wissen wie nutzen darf. Da der Angreifer nach wie vor ein grundlegendes Interesse hat, seinen Angriff ungehindert durchführen zu können, wird er nun zumindest versuchen, die Ableitung von Gegenmaßnahmen durch das System selbst zu beeinflussen bzw. zu unterbinden. Als Konsequenz hieraus ergibt sich, dass für die Auswertung der aufgezeichneten Informationen eine Instanz zu erschaffen ist, die selbst nicht zu dem potentiell kompromittierten System gehört und somit tatsächlich nicht kompromittierbar ist: Der Security-Überwacher. Diese Sicherheitsinstanz kann bzw. sollte sogar physisch in dem potentiell kompromittierten IT-System angesiedelt sein, damit eventuell abzuleitende Maßnahmen direkt ausgeführt werden können. Obgleich sich die Zugriffsweite auf die Logdaten theoretisch idealerweise nach dem zu untersuchenden Tatbestand orientiert, ist es für die Lösung des hier beschriebenen Sachverhaltes sinnvoll und darüber hinaus auch vollkommen ausreichend, dieser unabhängigen Instanz lediglich lesenden Zugriff einzuräumen. Diese Restriktion hat den Vorteil, dass die Logdaten hierüber seitens der Sicherheitsinstanz nicht verändert werden können. Sie hat allerdings auch den Nachteil, dass die Logdaten eine ebenfalls unabhängige und nicht kompromittierbare Zugriffsverwaltung benötigen! Als eine mögliche praktische Realisierung bietet sich die Verwendung von Zertifikaten und deren Einbettung in Hardware an.

Die Sicherheitsinstanz kann auf dem Security Core der in Hildebrandt et al. eingeführten Architektur laufen und in das Betriebssystem eines IT-sicheren Rechners integriert werden [HPK10].

Es gibt einen weiteren wichtigen Aspekt, der in diesem Zusammenhang zu berücksichtigen ist: Die Unterscheidung zwischen automatisierten Zugriffen der Sicherheitsinstanz und Zugriffen durch einen menschlichen Benutzer. Während erstere technisch zuverlässig durch eine Sicherheitsinstanz reglementiert werden kann, ist letztere von wirksamen, ergänzenden organisatorischen Vorkehrungen abhängig. Das bedeutet konkret, dass ein in diesem Zusammenhang möglicher Missbrauch auf alleiniger Basis einer sicheren Rechnerarchitektur nicht ausgeschlossen werden kann!

3.4 Auswertung der aufgezeichneten Inhalte: Was soll wann verfügbar sein?

Während der Fokus in den vorangegangenen zwei Abschnitten auf den grundsätzlichen Eigenschaften einer Aufzeichnungsinanz, sprich dem eigentlichen Sammeln und Aufbewahren von Informationen lag, steht nun die Frage der Auswertung derselben im Vordergrund. Das Kernproblem besteht darin, dass ein Auswertungsschema definiert werden muss, welches alle zuvor als sicherheitskritisch definierten Ereignisse aus den gewonnenen Informationen extrahiert bzw. Hinweise auf solche Ereignisse findet. Um tatsächlich eine wirksame Abwehr von Bedrohungen im Rechner im Rahmen der gewünschten incident-response-Architektur ermöglichen zu können, müssen die hieraus resultierenden Ergebnisse bzw. Entscheidungen zur

Laufzeit zur Verfügung stehen. Es ist deutlich erkennbar, dass die Realisierung einer solchen Auswertungsfunktionalität in mehrerlei Hinsicht nicht trivial ist:

- Erstens ist die zugrundeliegende Entscheidungslogik höchst komplex,
- zweitens muss sichergestellt sein, dass die Irrtumswahrscheinlichkeit, d.h. die Anzahl der false positives (α -Fehler) und false negatives (β -Fehler) unter einem bestimmten Signifikanzniveau liegen,
- drittens muss das Mengenproblem der anfallenden Logdaten bewältigt werden,
- viertens muss der gesamte Entscheidungsprozess mehrfach parallel ablaufen können,
- fünftens muss die Verarbeitung der Entscheidungsprozesse hochgradig performant sein und die Ergebnisse nahezu zeitgleich zur Verfügung stehen und
- sechstens muss die Initiierung von Maßnahmen im kompromittierten System durchgeführt werden können.

Mit der Erfüllung dieser Anforderungen löst sich nicht nur das Kernproblem der Auswertung zur Laufzeit, sondern es werden gleichermaßen die Bedürfnisse nachgelagerter Auswertungen bedient.

3.5 Ableitungen von Maßnahmen: Was soll wann passieren?

Die Frage, welche Aktionen bei bestimmten Ergebnissen automatisiert angestoßen werden sollen und wenn ja, zu welchem Zeitpunkt, ist generisch nicht zu beantworten. Der Grund hierfür liegt in der spezifischen Wirkung sowie den notwendigen Voraussetzungen individueller Angriffsvektoren und damit auch den unterschiedlichen Abwehrmechanismen. Spätestens zu diesem Zeitpunkt wird deutlich, dass eine statisch angelegte incident-response-Architektur wie z.B. eine Implementierung auf Basis von Angriffssignaturen schnell an ihre Grenzen stößt. Vielmehr wird man sich bei der Lösung vermutlich mit einem dynamisch handelnden, lernenden Ansatz auseinandersetzen müssen, um Bedrohungen tatsächlich wirksam unterbinden zu können.

Neben dieser Frage ergibt sich noch ein anderer Aspekt: Welche Instanz stößt diese Aktionen an? Sinnvollerweise darf diese Instanz nicht mit der auswertenden Instanz auf Zugriffsebene identisch sein, da sonst ein weiteres Kompromittierungspotential eingeführt wird. Praktisch bedeutet dies, dass beim Design des Security Cores auf eine entsprechende Trennung zu achten ist.

3.6 Referenzzeit: Wie und wo wird die Referenzzeit festgelegt?

Eigentlich sollte in der Theorie das zu entwerfende Sicherheitssystem zum jetzigen Zeitpunkt hinsichtlich seiner Anforderungen vollständig und konsistent beschrieben sein, d.h. die Möglichkeiten der Manipulation durch den Angreifer sollten faktisch nicht mehr gegeben sein. Allerdings gibt es noch einen Punkt, welcher der Voll-

ständigkeit halber erwähnt werden muss, weil er dem Angreifer nützlich ist und damit eine forensische Analyse negativ beeinflussen kann: Die Referenzzeit. Ohne auf die unzähligen Veröffentlichungen bzw. die dort beschriebenen Problemstellungen von zeitbasierenden Angriffsszenarien an dieser Stelle detailliert einzugehen, kann festgehalten werden, dass eine integere Referenzzeit in einem bekannten Format vorhanden sein muss, welche nicht manipulierbar sein darf. Eine praktische Implementierung könnte hier sein, die Referenzzeit von einem externen Zeitsystem zu beziehen.

4 Schlussfolgerung

Die Analyse hat gezeigt, dass es sinnvoll ist, eine IT-forensische Informationssammlung bereits zur Laufzeit auf Basis eines definierten, inhaltlichen Schemas zu optimieren, d.h. nur diejenigen Informationen zu verwenden, welche eine echte Aussagekraft haben und tatsächlich zu einem späteren Zeitpunkt benötigt werden. Die Einführung des Konzepts der Haltbarkeitsspanne liefert hierzu einen konkreten Lösungsansatz.

Zu dem Themenkomplex der Auswertungsfunktionalität bedarf es noch weiterer, intensiver Forschung. Insbesondere das Design und die Realisierung sind nach heutigem Kenntnisstand nicht trivial. Ein weiteres wichtiges Gebiet mit erheblichem Klärungspotential stellt die automatische incidence-reponse-Fähigkeit des zu entwickelnden Sicherheitssystems dar. Neben den technischen Prämissen und Abhängigkeiten muss vor allem die besondere rechtliche Situation umfänglich gewürdigt werden.

5 Quellen

- [HPK10] Klaus Hildebrandt, Igor Podebrad, and Bernd Klauer. A computer architecture with hardwarebased malware detection. Availability, Reliability and Security, International Conference on, 0:41–45, 2010.
- [PFMA09] Nick L. Petroni, Jr. Timothy Fraser, Jesus Molina, and William A. Arbaugh. Copilot – a coprocessor-based kernel runtime integrity monitor. www.usenix.org/events/seco4/tech/full_papers/petroni/petroni_html/main.html, 02.04.2009.
- [PHK09] Igor Podebrad, Klaus Hildebrandt, and Bernd Klauer. List of criteria for a secure computer architecture. Emerging Security Information, Systems, and Technologies, The International Conference on, 0:76–80, 2009.
- [Rut] Joanna Rutkowska. Security challenges in virtualized environments. <http://www.invisiblethings.org/papers/security%20Challenges%20in%20Virtualized%20Enviroments.pdf>.

- [Ruto6] Joanna Rutkowska. Introducing stealth malware taxonomy. <http://www.invisiblethings.org/papers/malware-taxonomy.pdf>, Nov. 2006.
- [Sre05] J.C. Sremack. Investigating real-time system forensics. In Security and Privacy for Emerging Areas in Communication Networks, 2005. Workshop of the 1st International Conference on, pages 25–32, Sept. 2005.

Das

FAT Dateisystem

von **Stefan Humml**

Abstract

Um im Rahmen der IT-Forensik korrekte, nachvollziehbare und einwandfreie Analysen durchführen zu können, ist ein sehr gutes Verständnis für Dateisysteme unabdingbar, da sie das Fundament bilden um Informationen auf Datenträgern zu speichern. Diese Arbeit vermittelt diese Grundlagen für das FAT-Dateisystem, das insbesondere im Umfeld mobiler Speicherlösungen wie z.B. USB-Sticks und Speicherkarten für Digitalkameras eingesetzt wird. Das Dateisystem wird hierzu zunächst beschrieben um darauf aufbauend forensische Ansätze wie beispielsweise das Wiederherstellen gelöschter Dateien oder die Analyse von Zeitstempeln unter Beachtung vorhandener Einschränkungen zu erläutern.

Inhaltsverzeichnis

| | |
|---|----|
| 1 Zielsetzung und Vorgehensweise | 23 |
| 1.1 Anmerkungen | 23 |
| 2 Das FAT-Dateisystem | 24 |
| 2.1 Strukturen | 24 |
| 2.1.1 Überblick | 24 |
| 2.1.2 Layout | 26 |
| 2.1.3 Boot-Sector | 26 |
| 2.1.4 Die FSINFO-Struktur | 27 |
| 2.1.5 Directory-Entries | 28 |
| 2.1.6 Lange Dateinamen | 29 |
| 2.1.7 Die File-Allocation-Table | 30 |
| 2.2 Zusammenhänge | 31 |
| 2.2.1 Verzeichnisse | 31 |
| 2.2.2 Cluster-Ketten | 32 |
| 3 Analyse | 33 |
| 3.1 Datum und Uhrzeit | 34 |
| 3.2 Wiederherstellen gelöschter Dateien | 36 |
| 3.3 Schlupfspeicher | 37 |
| 3.4 Einschränkungen | 38 |
| 4 Fazit und Ausblick | 39 |
| 5 Quellen | 40 |

Abbildungsverzeichnis

| | |
|--|----|
| Abbildung 1: Zusammenhang Sektor und Cluster (Beispiel: 1024 Bytes bzw. 2 Sektoren pro Cluster) | 3 |
| Abbildung 2: Zusammenhang der FAT-Basisstrukturen | 4 |
| Abbildung 3: Physikalisches Layout FAT | 5 |
| Abbildung 4: Beispiel einer Verzeichnisstruktur | 10 |
| Abbildung 5: Beispiel Cluster-Kette | 11 |
| Abbildung 6: RAM- und Drive-Slack eines FAT-Dateisystems (Clustergröße: 2048 Bytes) | 16 |

Tabellenverzeichnis

| | |
|---|----|
| Tabelle 1: FAT32 Boot-Sector | 6 |
| Tabelle 2: FSINFO-Struktur | 7 |
| Tabelle 3: Struktur eines Directory-Entries | 7 |
| Tabelle 4: Directory-Entry Attribute | 8 |
| Tabelle 5: Struktur eines LFN-Eintrags | 8 |
| Tabelle 6: Aufbau der FAT-Datumsstempel | 13 |
| Tabelle 7: Aufbau eines FAT-Zeitstempels | 13 |

Abkürzungsverzeichnis

| | |
|--------|--|
| ASCII | American Standard Code for Information Interchange |
| BIOS | Basic Input Output System |
| BPB | Bios Parameter Block |
| DOS | Disk Operating System |
| FAT | File Allocation Table |
| FSINFO | File System Information |
| HPSF | High Performance File System |
| LFN | Long Filename |
| MAC | Modified, Accessed, Created |
| MS | Microsoft |
| NTFS | New Technology File System |
| RAM | Random Access Memory |
| USB | Universal Serial Bus |

1 Zielsetzung und Vorgehensweise

Dateisysteme bilden die logische Grundlage, um Informationen auf Datenträgern zu speichern und zu verwalten. Ein Verständnis für Dateisysteme ist daher von fundamentaler Bedeutung, um eine forensische Analyse durchführen zu können. Das genaue Wissen um den Aufbau und die Arbeitsweise eines Dateisystems ermöglicht es, exakt, nachvollziehbar und vollständig alle verfügbaren Informationen eines mit diesem Dateisystem beschriebenen Datenträgers zu extrahieren.¹

Ziel der Arbeit ist die systematische Beschreibung des Dateisystems FAT (File Allocation Table), das im Umfeld von Microsoft-Betriebssystemen sowie mobilen Datenträgern wie USB-Speichern und Massenspeichern von Digitalen Kameras zum Einsatz kommt². Neben der Beschreibung des Dateisystems werden weiterhin Informationen zur forensischen Analyse erörtert.

Kapitel 2 beschreibt das FAT-Dateisystem und stellt zunächst elementare Datenstrukturen vor. Im Weiteren werden die Zusammenhänge dieser Datenstrukturen erläutert und somit die Funktionsweise des Dateisystems erklärt. Kapitel 3 beschreibt Möglichkeiten und Einschränkungen der forensischen Analyse eines FAT32-Dateisystems anhand der zuvor erläuterten Strukturen und Zusammenhänge.

Die Arbeit setzt hierbei grundsätzliche Kenntnisse von Rechnerarchitekturen, Betriebssystemen und Computerhardware voraus.

1.1 Anmerkungen

Primäre Quelle für die Beschreibung des FAT-Dateisystems in ein Microsoft Whitepaper ([Microsoft 2000]). Es existiert keine weitere Literatur von Microsoft, die das Dateisystem ähnlich vollständig beschreibt.

Kapitel 2 beschränkt sich auf die Beschreibung des FAT32-Dateisystems. Vorhergehende Versionen beruhen auf den gleichen Prinzipien, verwenden jedoch an einigen Stellen andere Datentypen³. Dieses Vorgehen wurde gewählt, da zum Zeitpunkt der Erstellung dieser Arbeit fast ausschließlich FAT32 eingesetzt wird⁴.

Alle Zahlenangaben dieser Arbeit, insbesondere innerhalb von Tabellen, beziehen sich auf das Dezimalsystem. Hexadezimale Zahlenangaben werden durch vorangestelltes „0x“ kenntlich gemacht.

1 [Carrier 2005, Vorwort, S. xvi]

2 [Russinovich 2005, S. 689 f], [Carrier 2005, S. 211]

3 Insbesondere die Einträge der FAT-Struktur, die sich in der Namensgebung des Dateisystems (FAT12, FAT16, FAT32) ausdrücken, besitzen unterschiedliche Größen. Vgl. hierzu [Carrier 2005, S. 212].

4 [Steel 2006, S. 66]

2 Das FAT-Dateisystem

Das FAT-Dateisystem hat seine Wurzeln in den späten siebziger Jahren des zwanzigsten Jahrhunderts und wurde von Microsoft im Rahmen des MS-DOS⁵ Betriebssystems für den Einsatz auf Disketten entwickelt. Im Laufe der Zeit wurden neue Versionen geschaffen, die jeweils wachsenden Anforderungen wie beispielsweise größer werdenden Datenträgern angepasst wurden⁶. Aus diesen Erweiterungen und Anpassungen entstanden verschiedene Versionen des Dateisystems⁷:

- FAT12
- FAT16
- FAT32

Dieses Kapitel beschreibt Datentypen und Strukturen und erläutert anschließend die Funktionsweise des FAT32-Dateisystems⁸.

2.1. Strukturen

Dieser Abschnitt gibt einen Überblick über die für FAT32 wichtigen Strukturen und Datentypen. Weiterhin werden relevante Begrifflichkeiten definiert.

2.1.1 Überblick

FAT verwendet, um Daten auf der Festplatte zu speichern, sog. *Cluster*; dabei handelt es sich um zusammenhängende Reihen von Sektoren einer Festplatte. Mögliche Clustergrößen liegen zwischen 512 Bytes und 32 KB, wobei der Wert generell als eine Potenz von Zwei ausgedrückt werden muss⁹. Die folgende Abbildung stellt exemplarisch den Zusammenhang zwischen Sektoren und Clustern da.

| | | | | | | | | | |
|-------|-----------|----------|-----------|----------|-----------|----------|------|-----|-----|
| Byte: | 0 | 512 | 1024 | 1536 | 2048 | 2560 | 3072 | ... | ... |
| | Sektor 1 | Sektor 2 | Sektor 3 | Sektor 4 | Sektor 5 | Sektor 6 | ... | ... | |
| | Cluster 1 | | Cluster 2 | | Cluster 3 | | | | |

ABBILDUNG 1: ZUSAMMENHANG SEKTOR UND CLUSTER (BEISPIEL: 1024 BYTES BZW. 2 SEKTOREN PRO CLUSTER)

5 Microsoft Disk Operating System

6 [Microsoft 2000, S.1]

7 [Microsoft 2000, S.1], [Carrier 2005, S. 211], [Sammes 2007, S. 191]

8 Vgl. Abschnitt 1.1 – Anmerkungen

9 [Carrier 2005, S. 221–222]

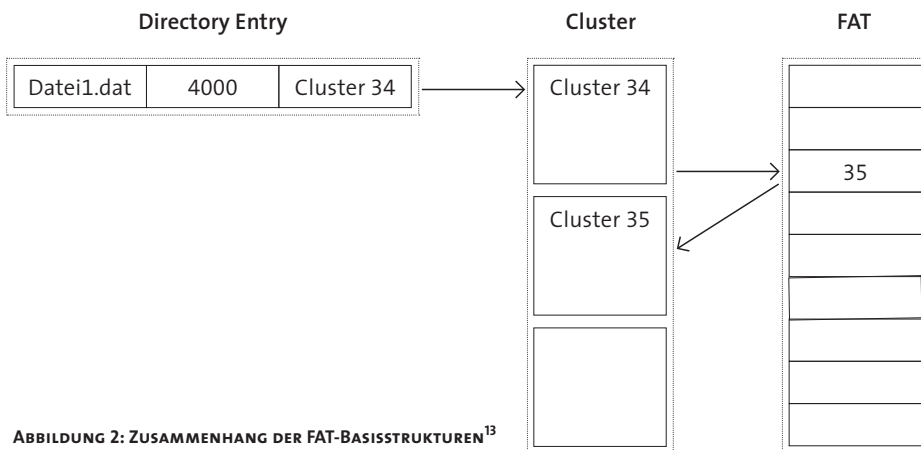
Bei Clustern handelt es sich um die kleinsten adressierbaren Datenbereiche des Dateisystems¹⁰. Weiterhin ist jedem Cluster auf der Festplatte genau ein Eintrag innerhalb der File-Allocation-Table zugewiesen.

Innerhalb des FAT-Dateisystems wird jede Datei und jedes Verzeichnis durch einen *Directory-Entry* (dt. Verzeichnis-Eintrag) beschrieben. Dieser Eintrag enthält Informationen wie

- Name der Datei
- Größe in Bytes
- Nummer des ersten Clusters
- Weitere Metainformationen.¹¹

Sollte eine Datei aufgrund ihrer Größe mehrere Cluster in Anspruch nehmen, so ist dies in der File Allocation Table gekennzeichnet. In diesem Fall enthält der entsprechende FAT-Eintrag die Adresse des nächsten Clusters, in dem Daten der Datei gespeichert sind¹². Auf diese Weise kann die FAT ebenfalls verwendet werden, um den Allokationsstatus eines Clusters zu bestimmen: lautet ein Eintrag 0x00, so ist der korrespondierende Cluster nicht belegt und steht für neue Einträge und Dateien zur Verfügung.

Die folgende Abbildung verdeutlicht den Zusammenhang zwischen den beschriebenen Strukturen:



10 Der erste Cluster eines FAT-Systems ist Cluster Nummer 2. Cluster null und eins existieren nicht. Dies muss bei Analysen berücksichtigt werden. Vgl. [Carrier 2005, S. 221], [Sammes 2007, S. 191]

11 [Carrier 2005, S. 227–228]

12 Hier ist ein wesentlicher Unterschied der verschiedenen FAT-Versionen zu erkennen. Die Größe der FAT-Einträge entspricht bei FAT12 zwölf Bits, bei FAT16 16 Bits und bei FAT32 32 Bits.

13 Vgl. [Carrier 2005, S. 212]

2.1.2 Layout

Das FAT32-Dateisystem teilt eine Partition (oder einen Datenträger) in drei Bereiche. Zunächst befindet sich ein reservierter Bereich am Anfang der Partition; dieser enthält vitale Informationen zu Größe und Aufbau des nachfolgenden Dateisystems. An den reservierten Bereich schließt sich der FAT-Bereich an, der die *File-Allocation-Table* und eine Kopie derselben enthält. Darauf folgt der *Daten-Bereich*, der die *Cluster* enthält und somit *Directory-Entries* und *Datei-Inhalte* aufnimmt. Abbildung 3 beschreibt dieses Layout.

| | | |
|----------------------|-------------|-------------------------|
| reservierter Bereich | FAT Bereich | Daten (Cluster) Bereich |
|----------------------|-------------|-------------------------|

ABBILDUNG 3: PHYSIKALISCHES LAYOUT FAT ¹⁴

2.1.3 Boot-Sector

Der *Boot-Sector*, Microsoft spricht von dem sog. Bios Parameter Block (BPB) ¹⁵, befindet sich in dem ersten Sektor des Datenträgers; somit liegt er im *reservierten Bereich* ¹⁶. Hier sind wesentliche Informationen über Lage und Layout des nachfolgenden FAT-Systems beschrieben. Diese Informationen werden von Seiten des Betriebssystems benötigt, um auf Inhalte des Dateisystems zugreifen zu können. Die genaue Bedeutung der einzelnen Einträge des *Boot-Sectors* können Tabelle 1 entnommen werden ¹⁷.

| Offset | Größe | Beschreibung |
|--------|-------|---|
| 0 | 3 | Sprung-Anweisung (Assembler-Instruktion) um den Boot-Code zu erreichen. |
| 3 | 8 | OEM-Name (ASCII ¹⁸). z.B. „MSWIN4.1“ |
| 11 | 2 | Bytes pro Sektor. Mögliche Werte: 512, 1024, 2048, 4096 |
| 13 | 1 | Sektoren pro Cluster |
| 14 | 2 | Anzahl reservierte Sektoren (Reservierter Bereich) |
| 16 | 1 | Anzahl FAT-Strukturen. Im Regelfall zwei, kann jedoch abweichen. |
| 17 | 2 | Anzahl Root-Directory-Entries. Null für FAT32. |

¹⁴ Vgl. [Carrier 2005, S. 213]

¹⁵ [Microsoft 2000, S. 7] und [Carrier 2005, S. 213]

¹⁶ Vgl. *Abschnitt 2.1.2*

¹⁷ Bis einschließlich Byte 35 ist das Layout aller FAT-Versionen gleich. Die hier beschriebenen Strukturen ab Offset 36 beziehen sich auf FAT-32-Systeme.

¹⁸ *American Standard Code for Information Interchange*, eine 7 Bit Zeichenkodierung.

| | | |
|-----|-----|---|
| 19 | 2 | Anzahl Sektoren auf dem Volume. Null (nicht verwendet) für FAT32 ¹⁹ |
| 21 | 1 | Medientyp. oxF8 für Standard-Volumes (z.B. Festplatten), oxFo für entfernbare Datenträger wie beispielsweise USB ²⁰ -Sticks. |
| 22 | 2 | Anzahl verwendeter Sektoren für eine FAT. Null für FAT32. |
| 24 | 2 | Sektoren pro Track (Festplattengeometrie) |
| 26 | 2 | Anzahl Köpfe (Festplattengeometrie) |
| 28 | 4 | Anzahl Sektoren vor Beginn der Partition |
| 32 | 4 | Anzahl Sektoren auf dem Volume (vgl. Offset 19) |
| 36 | 4 | Anzahl Sektoren für eine FAT |
| 40 | 2 | Beschreibt, wie mit mehreren FAT-Strukturen umzugehen ist ²¹ |
| 42 | 2 | Versionsnummer des Dateisystems |
| 44 | 4 | Adresse (Sektor) des Root-Directory-Clusters |
| 48 | 2 | Adresse (Sektor) der FSINFO-Struktur |
| 50 | 2 | Adresse (Sektor) Kopie des Boot-Sektors |
| 52 | 12 | Reserviert |
| 64 | 1 | BIOS ²² Interrupt 13 Datenträgernummer |
| 65 | 1 | Nicht verwendet |
| 66 | 1 | Signatur (ox29) |
| 67 | 4 | Seriennummer der Festplatte |
| 71 | 11 | Volume Name (Bezeichnung) (ASCII) |
| 82 | 8 | Dateisystemlabel (i.d.R. „FAT32“) |
| 90 | 420 | Nicht verwendet |
| 510 | 2 | Signatur (oxAA55) |

TABELLE 1: FAT32 BOOT-SECTOR²³

2.1.4 Die FSINFO-Struktur

Jedes FAT32-Dateisystem enthält eine sog. FSINFO-Struktur²⁴, um dem Betriebssystem Informationen zur Verfügung zu stellen, die sonst aus anderen Daten berechnet werden müssten²⁵. Die Notwendigkeit einer solchen Struktur entstand aus der Größe der Datenträger, die FAT32 ermöglicht. Beispielsweise ist es zu aufwendig, die freie Größe eines Dateisystems bei jeder Anfrage zu errechnen²⁶.

19 Da es sich um einen 16-Bit-Wert handelt waren die Dateisysteme FAT12 und FAT16 in der Größe begrenzt. FAT32 verwendet diesen Eintrag nicht mehr.

20 Universal Serial Bus

21 Siehe [Microsoft 2000, S. 12] und [Carrier 2005, S. 256] für die Belegung der einzelnen Bits.

22 Basic Input Output System

23 Vgl. [Carrier 2005, S. 254] und [Microsoft 2000, S. 9]

24 *File-System-Information* Struktur

25 [Carrier 2005, S. 259]

26 [Microsoft 2000, S. 21]

Da, wie beschrieben, jede der in FSINFO festgehaltenen Informationen erneut errechnet werden kann, handelt es sich dabei um keine unverzichtbaren Daten²⁷. Die folgende Tabelle stellt mögliche Inhalte der FSINFO-Struktur vor.

| Offset | Größe | Beschreibung |
|--------|-------|---|
| 0 | 4 | Signatur (0x41615252 – „AaRR“) |
| 4 | 480 | Für spätere Verwendung reserviert. Sollte jedoch mit 0x00 initialisiert werden. |
| 484 | 4 | Signatur (0x61417272 – „aArr“) |
| 488 | 4 | Die zuletzt bekannte Anzahl freier Cluster. 0xFFFFFFFF falls nicht bekannt. |
| 492 | 4 | Nächster freier Cluster |
| 496 | 12 | Für spätere Verwendung reserviert. Sollte jedoch mit 0x00 initialisiert werden. |
| 508 | 4 | Signatur (0xAA550000) |

TABELLE 2: FSINFO-STRUKTUR²⁸

2.1.5 Directory-Entries

Ein *Directory-Entry* enthält *Metainformationen* zu einer Datei. Jede Datei und jedes Verzeichnis basiert auf einem solchen Eintrag²⁹. Directory-Entries sind im Daten-Bereich³⁰ des Dateisystems angesiedelt und können mit Hilfe der Clusternummer adressiert werden³¹. Sie sind nicht nummeriert; daraus folgt, dass der einzige Weg sie zu lokalisieren darin besteht, ihren Standort ausgehend vom Root-Directory zu ermitteln. Dazu ist eine vollständige Pfadangabe unerlässlich.³²

Der genaue Aufbau eines Directory-Entries ist der folgenden Tabelle zu entnehmen:

| Offset | Größe | Beschreibung |
|--------|-------|---|
| 0 | 11 | Dateiname (kurz; 8.3) |
| 11 | 1 | Datei-Attribute |
| 12 | 1 | Reserviert |
| 13 | 1 | Erstellungs-Zeit (Zehntel Sekunden) |
| 14 | 2 | Uhrzeit der Datei-Erstellung (Stunden, Minuten, Sekunden) |

27 Zur physikalischen Lokalisierung der FSINFO-Struktur siehe *Tabelle 1, Offset 48*

28 Vgl. [Carrier 2005, S. 259] und [Microsoft 2000, S. 21]

29 [Carrier 2005, S. 227]

30 Vgl. Abschnitt 2.1.2 – *Layout*

31 Dennoch kann das Root-Directory über eine Sektor-Nummer adressiert werden. Vgl. *Tabelle 1, Offset 44*

32 [Carrier 2005, S. 227]

| | | |
|----|---|---|
| 16 | 2 | Erstellungs-Datum |
| 18 | 2 | Datum des letzten Zugriffs |
| 20 | 2 | Ersten zwei Bytes (high word) der Adresse des ersten Clusters |
| 22 | 2 | Uhrzeit des letzten Schreibzugriffs (Stunden, Minuten, Sekunden) |
| 24 | 2 | Datum des letzten Schreibzugriffs |
| 26 | 2 | Letzten zwei Bytes (low word) der Adresse des ersten Clusters |
| 28 | 4 | Dateigröße (null für Verzeichnisse) |

TABELLE 3: STRUKTUR EINES DIRECTORY-ENTRIES³³

Das erste Byte (Offset 0) der aufgezeigten Struktur wird verwendet, um den Allokationsstatus des Eintrags zu ermitteln. Ist hier der Wert 0x00 oder 0xE5 zu finden, ist der Eintrag frei und kann bei Bedarf überschrieben werden. In jedem anderen Fall stellt dieses Byte den ersten Buchstaben des Dateinamens dar. Die Angabe erfolgt i.d.R. in ASCII. Es sind jedoch andere Darstellungen möglich.³⁴

Die Attribute eines *Directory-Entry* bestimmen, wie mit dem Inhalt dieses Eintrags verfahren wird. Sie werden durch ein einzelnes Byte dargestellt (Offset 11) und können anhand ihrer Bitmaske ermittelt werden:

| Flags ⁶ | Beschreibung |
|--------------------|-------------------------|
| 0000 0001 (0x01) | Nur lesen |
| 0000 0010 (0x02) | Versteckt |
| 0000 0100 (0x04) | Systemdatei |
| 0000 1000 (0x08) | Datenträger-Bezeichnung |
| 0000 1111 (0x0F) | Langer Dateiname |
| 0001 0000 (0x10) | Verzeichnis |
| 0010 0000 (0x20) | Archiv |

TABELLE 4: DIRECTORY-ENTRY ATTRIBUTE³⁵

2.1.6 Lange Dateinamen

Wie Tabelle 3 (Offset 0) zu entnehmen ist, können in einer normalen Directory-Entry-Struktur lediglich kurze Dateinamen im sog. 8.3-Format³⁶ gespeichert werden.

³³ Vgl. [Carrier 2005, S. 262] und [Microsoft 2000, S. 23]

³⁴ [Carrier 2005, S. 262]

³⁵ Vgl. [Carrier 2005, S. 263]

³⁶ Das 8.3-Format leitet sich aus der Separierung von Dateinamen und -erweiterung ab. Es stehen acht Zeichen für den Namen und drei Zeichen für die Erweiterung zur Verfügung.

FAT32 beherrscht jedoch längere Dateinamen. Zu diesem Zweck wurde eine weitere Struktur eingeführt. Dabei handelt es sich um sog. *Long File Name Directory Entries* (LFN Entries), deren Struktur in der folgenden Tabelle dargestellt wird.³⁷

| Offset | Größe | Beschreibung |
|--------|-------|--|
| 0 | 1 | Sequenznummer und Allokationsstatus ⁷ |
| 1 | 10 | Buchstaben 1–5 des Dateinamens (Unicode) |
| 11 | 1 | Datei-Attribut (muss oxof sein) |
| 12 | 1 | Reserviert |
| 13 | 1 | Checksumme ⁸ |
| 14 | 12 | Buchstaben 6–11 des Dateinamens (Unicode) |
| 26 | 2 | Reserviert |
| 28 | 4 | Buchstaben 12–13 des Dateinamens (Unicode) |

TABELLE 5: STRUKTUR EINES LFN-EINTRAGS³⁸

Um Abwärtskompatibilität mit alten DOS und Windows-Versionen zu erreichen, werden LFN-Einträge wie normale Directory-Entries verwaltet. Jedoch haben sie ein spezielles Dateiattribut (oxof)³⁹, das von allen vorherigen Dateisystemtreibern ignoriert wird⁴⁰. Ist ein Eintrag für lange Dateinamen vorhanden, so steht dieser generell direkt vor dem herkömmlichen Directory-Entry. Jeder LFN-Eintrag ermöglicht das Speichern von 13 Unicode-Buchstaben. Sollte für das Schreiben des Dateinamens mehr Platz benötigt werden, können mehrere LFN-Einträge allokiert werden. Ist dies der Fall, werden die LFN-Einträge in umgekehrter Reihenfolge gespeichert. Die Anzahl verwendeter LFN-Einträge für eine Datei kann mit Hilfe der Sequenznummer (Offset 0) bestimmt werden.⁴¹

2.1.7 Die File-Allocation-Table

Die File-Allocation-Table stellt eine zentrale Struktur des Dateisystems dar. Sie besteht aus einzelnen, 32 Bit breiten Einträgen⁴² und enthält keine weiteren Informationen. Jedem Cluster innerhalb des Datenbereiches wird genau ein Eintrag in der File-Allocation-Table zugewiesen, woraus eine Eins-zu-eins-Beziehung entsteht. Auf diese Weise ist es möglich, den Status eines Clusters durch den Eintrag in der FAT zu ermitteln.

37 [Carrier 2005, S. 267] und [Microsoft 2000, S. 26]

38 Vgl. [Carrier 2005, S. 267] und [Microsoft 2000, S. 26–27]

39 Vgl. „Tabelle 3: Struktur eines Directory-Entries“ und „Tabelle 5: Struktur eines LFN-Eintrags“.

Beide Strukturen definieren an Offset 11 das Feld Datei-Attribut, wobei der Inhalt für einen LFN-Eintrag festgelegt ist.

40 [Microsoft 2000, S. 26]

41 [Carrier 2005, S. 267]

42 FAT32 verwendet 32 Bit breite Strukturen. Siehe hierzu die Einführung in Kapitel 2.1.1 – Überblick

Der Inhalt eines FAT-Eintrags kann in drei Kategorien eingeteilt werden:

1. Der korrespondierende Cluster ist nicht belegt. Eintrag: 0x00000000
2. Der Cluster gehört zu einer Datei und es existiert ein weiterer Cluster, der zu dieser Datei gehört. Eintrag: Nummer des nächsten Clusters
3. Der Cluster gehört zu einer Datei und es existiert kein weiterer Cluster, der zu dieser Datei gehört. Eintrag: Ende-Markierung⁴³.

Die elementare Bedeutung der FAT sowie ihre Rolle im Aufbau des Dateisystems wird im nächsten Abschnitt beschrieben.

2.2 Zusammenhänge

Dieser Abschnitt stellt die Zusammenhänge zwischen den in Abschnitt 2.1 eingeführten Datentypen her. Auf diese Weise wird erläutert, wie das Dateisystem aufgebaut ist und Dateizugriffe funktionieren.

2.2.1 Verzeichnisse

Verzeichnisse werden durch einen speziellen Typ des Directory-Entries beschrieben⁴⁴. Sie sind demzufolge wie normale Dateien zu behandeln; der Inhalt eines Verzeichnisses wird, wenn Bedarf besteht, in Form von Cluster-Ketten aufgebaut.⁴⁵

Ist ein Cluster als Verzeichnis allokiert worden, so enthält er eine Reihe Directory-Entries, die den Inhalt des Verzeichnisses darstellen.

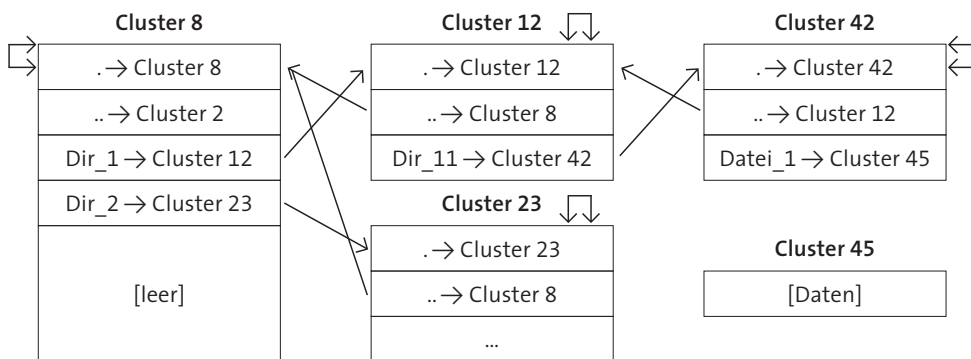


ABBILDUNG 4: BEISPIEL EINER VERZEICHNISSTRUKTUR

43 In FAT32 wurde ein Wert größer 0x0FFF FFF8 festgelegt. [Carrier 2005, S. 260]

44 Attribut: 0x10

45 [Carrier 2005, S. 229 ff]. Zu Cluster-Kette siehe folgenden Abschnitt.

Abbildung 4 beschreibt eine exemplarische Verzeichnisstruktur unter Zuhilfenahme der FAT-Strukturen. Die durch die Abbildung skizzierte Struktur lässt sich folgendermaßen darstellen⁴⁶:

```
/
+Dir_1
|  |
|  + DIR_11
|    |
|    +Datei_1
|
+Dir_2
  |
  + ...
```

2.2.2 Cluster-Ketten

Wie in 2.1.5 beschrieben enthält ein Directory-Entry einen *Verweis* auf den ersten Cluster einer Datei und referenziert auf diese Weise den ersten Block des Dateiinhalts.

Ist die Datei jedoch größer als die Clustergröße des Dateisystems und kann somit nicht in einem Cluster gespeichert werden, werden weitere Cluster allokiert, und der Dateiinhalt sukzessiv in diese geschrieben.

Um auf den gesamten Inhalt der Datei zugreifen zu können wird folgendermaßen verfahren: Zunächst wird der entsprechende Directory-Entry gesucht. In diesem befindet sich eine Referenz auf den ersten Cluster der Datei; somit kann der Inhalt des ersten Clusters ausgelesen werden. Um den nächsten Cluster in Erfahrung zu bringen wird der FAT-Eintrag des Clusters ausgelesen. Ist dieser nicht mit einer Ende-Markierung versehen, referenziert der FAT-Eintrag den nächsten Cluster, der Inhalte der Datei speichert. Eine auf diese Weise abgearbeitete Reihe von Clustern nennt man Clusterkette (engl. *cluster chain*)⁴⁷.

46 Unter der Annahme, dass sich das Root(Basis)-Verzeichnis in Cluster 8 befindet.

47 Vgl. [Carrier 2005, S. 228]

Folgende Abbildung verdeutlicht die Zusammenhänge:

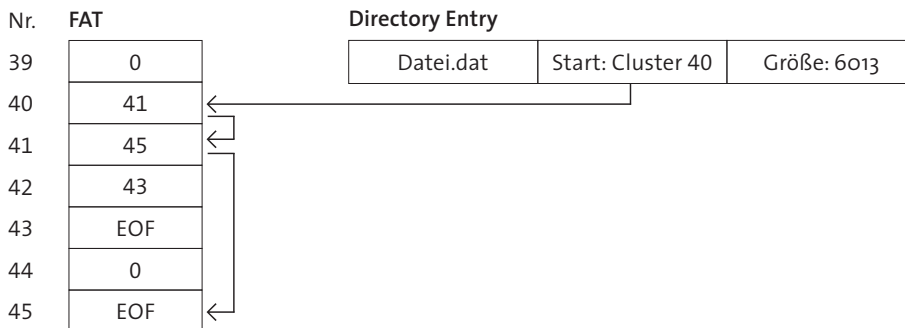


ABBILDUNG 5: BEISPIEL CLUSTER-KETTE⁴⁸

In der Abbildung wird die Datei „Datei.dat“ durch den Directory Entry als 6013 Byte lange Datei mit dem Startcluster 40 beschrieben. Mit Hilfe dieser Informationen ist das Betriebssystem in der Lage, die Datei vollständig auszulesen: zunächst wird der Cluster mit der Nummer 40 gelesen und als Inhalt der Datei interpretiert. Darauf folgt der Zugriff auf Element Nr. 40 der FAT, der angibt, dass der weitere Dateiinhalte in Cluster Nr. 41 zu finden ist. Der korrespondierende Eintrag in der FAT gibt Cluster 45 als nachfolgenden an. Nach Auslesen dessen Inhalts zeigt der Zugriff auf FAT-Eintrag Nr. 45, dass keine weiteren Einträge folgen.

3 Analyse

In diesem Kapitel werden die Möglichkeiten der forensischen Analyse eines FAT-Dateisystems beschrieben. Abschnitt 3.1 erläutert, wie Daten und Zeiten des Dateisystems zu interpretieren sind und welche Informationen daraus gewonnen werden können.

Aus forensischer Sicht ist das Wiederherstellen gelöschter Dateien ein weiterer wichtiger Faktor einer Untersuchung⁴⁹; entsprechende Möglichkeiten werden in 3.2 erläutert. Der darauf folgende Abschnitt beschreibt den sog. Schlupfspeicher, der ebenfalls als Quelle forensisch relevanter Informationen gilt⁵⁰.

Das Kapitel schließt mit einer Beschreibung der Einschränkungen, die bei der Analyse eines FAT Dateisystems zu beachten sind.

48 Vgl. [Carrier 2005, S. 229]

49 [Geschonnek 2004, S. 96–97]

50 [Carrier 2005, S. 187]

3.1 Datum und Uhrzeit

Im Rahmen forensischer Analysen ist es insbesondere relevant herauszuarbeiten, zu welchem Zeitpunkt eine Datei erstellt, gelesen oder verändert wurde⁵¹. Mit diesen Informationen können vermutete Tatzeiten ermittelt, Zugriffszeiten bestimmter Programme oder die Internetnutzung eines Verdächtigen nachvollzogen werden. Eine Analyse dieser Zeiten gilt als fundamentaler Bestandteil einer jeden forensischen Analyse⁵² und mündet in einem Zeitstrahl (engl. timeline), auf dem für die Untersuchung relevante Ereignisse chronologisch abgetragen werden⁵³.

Zu diesem Zweck werden Daten und Uhrzeiten, die einer Datei zugerechnet werden können, in drei Kategorien aufgeteilt, die Rückschlüsse über den Umgang mit dieser Datei zulassen⁵⁴:

1. Änderungszeitpunkt der Datei (engl.: modification time, kurz: m-time)
2. Zeitpunkt des letzten (lesenden) Zugriffs (engl.: access time, kurz: a-time)
3. Erstellungszeitpunkt der Datei (engl.: creation time, kurz: c-time)⁵⁵

Ein FAT-Dateisystem hält diese Zeitstempel in der Directory-Entry Struktur vor⁵⁶. Diese können jedoch nur richtig interpretiert werden, wenn bekannt ist, auf welche Weise sie vom Betriebssystem geschrieben werden. Die FAT-Spezifikation sieht sowohl für das Zeit- als auch das Datumsfeld 16-Bit Werte vor, die im Folgenden beschrieben werden⁵⁷.

Mit Hilfe der Datumsstempel können Daten zwischen dem 01.01.1980 und dem 31.12.2107 dargestellt werden:

| Bits | Bedeutung |
|------|---|
| 0–4 | Tag; mögliche Werte: 1–31 |
| 5–8 | Monat; mögliche Werte: 1–12 |
| 9–15 | Jahr. Die Zählung beginnt 1980; mögliche Werte: 0–127 |

TABELLE 6: AUFBAU DER FAT-DATUMSSTEMPEL⁵⁸

51 [Geschonnek 2004, S. 88]

52 [Steel 2006, S. 72]

53 [Geschonnek 2004, S. 151]

54 [Carrier 2005, S. 196], [Geschonnek 2004, S. 88]

55 Aufgrund der englischen Abkürzungen werden die Zeit- und Datumsstempel MAC-Zeiten (engl. MAC-times) genannt.

56 Vgl. *Tabelle 3: Struktur eines Directory-Entries, Offsets 13–20*

57 [Carrier 2005, S. 262–264], [Microsoft 2000, S. 25]

58 [Microsoft 2000, S. 25]

Folgende Tabelle zeigt den Aufbau der Zeitstempel eines FAT-Dateisystems. Hierbei ist zu bemerken, dass diese nicht sekundengenau sind, sondern aufgrund der Einschränkung des Sekundenanteils auf fünf Bit lediglich jede zweite Sekunde festhalten können⁵⁹.

| Bits | Bedeutung |
|-------|---|
| 0–4 | 2-Sekunden-Zähler; mögliche Werte: 0–29 |
| 5–10 | Minuten; mögliche Werte: 0–59 |
| 11–15 | Stunden; mögliche Werte: 0–23 |

TABELLE 7: AUFBAU EINES FAT-ZEITSTEMPELS⁶⁰

Mit Hilfe der Informationen aus den Tabellen 3, 6 und 7 können aus einem FAT-Dateisystem folgende Informationen gewonnen werden⁶¹:

- Datum und Uhrzeit der Dateierstellung⁶²
- Datum und Uhrzeit des letzten Schreibzugriffs
- Datum des letzten lesenden Zugriffs

Die gewonnenen Informationen über Zugriffszeitpunkte einer Datei müssen jedoch im Rahmen der gesamten forensischen Arbeit kritisch betrachtet werden und dürfen nicht als alleiniges Indiz einer Tat verwendet werden, da insbesondere bei FAT das Verändern dieser Zeiten als sehr einfach gilt⁶³. Beispielsweise ist dies durch Kopieren bzw. Verschieben der Datei über verschiedene Datenträger hinweg realisierbar. Weiterhin kann mit Hilfe eines sog. Disk-Editors, der einen direkten Zugriff auf die Festplatte zulässt, der Zeitstempel manuell verändert werden. Eine dritte Manipulationsmöglichkeit stellt die (temporäre) Veränderung der Systemzeit dar, die sich in Schreiboperationen des Dateisystems widerspiegelt und somit die reale Zugriffszeit verschleiert⁶⁴.

59 Vgl. hierzu auch [Carrier 2005, S. 263]

60 [Microsoft 2000, S. 25]

61 Wobei für die Genauigkeit des Sekundenanteils jeweils die oben genannte Einschränkung gilt.

62 Hierbei gilt die oben genannte Einschränkung des Sekundenteils. Jedoch trägt ein Directory-Entry für den Erstellungszeitpunkt auch den Zehntelsekundenanteil. [Microsoft 2000, S. 23]. Vgl. *Tabelle 3, „Offset 13“*.

63 [Carrier 2005, S. 236], [Geschonnek 2004, S. 88]

64 Dies setzt entsprechende Rechte des Benutzers im Betriebssystem voraus. Bei mobilen Geräten wie beispielsweise digitalen Kameras jedoch ist es in der Regel jedem Benutzer möglich, Datum und Uhrzeit zu verändern.

3.2 Wiederherstellen gelöschter Dateien

Die Tatsache, dass gelöschte Dateien auf einem FAT-Datenträger relativ leicht wieder hergestellt werden können, ist Computerbenutzern häufig nicht bekannt⁶⁵. Davon ausgehend nehmen Täter an, einmal gelöschte Dateien blieben dem Ermittler verborgen. Aus diesem Grund ist die forensische Analyse gelöschter Dateien von besonderer Bedeutung. Dieser Abschnitt der Arbeit stellt dar, wie gelöschte Dateien auf einem FAT-Datenträger wieder hergestellt werden können und beschreibt, in welchen Fällen dies möglich ist.

Um die Wiederherstellung einer Datei auf einem FAT-Datenträger nachzuvollziehen ist es zunächst notwendig, den Vorgang des Löschens zu beschreiben. Wird eine Datei gelöscht, werden folgende Schritte auf dem Dateisystem vorgenommen:

- Der entsprechende Verzeichnis-Eintrag (directory entry) wird modifiziert, indem der erste Buchstabe des Dateinamens durch 0xE5 markiert und somit der Eintrag zur weiteren Verwendung frei gegeben wird⁶⁶.
- Sämtliche der Datei zugehörigen Einträge in der File Allocation Table werden durch Null ersetzt und somit ebenfalls freigegeben.

Hierbei ist hervorzuheben, dass die allokierten und von der gelöschten Datei verwendeten Cluster innerhalb des Datenbereichs nicht überschrieben werden⁶⁷. Da diese jedoch während des Löschvorgangs in der FAT als frei markiert werden, ist es möglich, dass das Betriebssystem diese wieder verwendet und somit die Daten der alten Datei überschreibt. Aus diesem Grund ist es nicht garantiert, dass die Datei vollständig wiederhergestellt werden kann⁶⁸.

Ferner ist die Reihenfolge der Cluster, die mit Hilfe der File Allocation Table beschrieben wurde, nicht mehr nachvollziehbar. Da diese Information unwiederbringlich verloren ist, haben sich zwei Ansätze bzw. Algorithmen etabliert, um die Datei dennoch wiederherstellen zu können, wobei die Umsetzung des zweiten Ansatzes häufiger in der Lage ist, die Datei vollständig und korrekt wiederherzustellen:⁶⁹

- Beginnend mit dem ersten Cluster der Datei werden, auf Basis der weiterhin bekannten Dateigröße⁷⁰, die direkt nachfolgenden Cluster der Datei zugeordnet und entsprechend ausgelesen.
- Beginnend mit dem ersten Cluster der Datei werden, auf gleicher Datenbasis, alle nachfolgenden, *als frei markierten* Cluster der gelöschten Datei zugeordnet und ausgelesen.

65 [Geschonnek 2004, S. 96]

66 Vgl. Abschnitt 2.1.5 – *Directory-Entries*

67 [Steel 2006, S. 226-228]

68 [Steel 2006, S. 227], [Sammes 2007, S. 199]

69 [Carrier 2005, S. 247]

70 Die Adresse des ersten Clusters einer Datei sowie deren Größe kann dem als gelöscht markierten Directory Entry entnommen werden.

Um diese Tätigkeit nicht manuell durchführen zu müssen, gibt es eine Reihe von Werkzeugen, die automatisch versuchen, Dateien auf FAT-Datenträgern wiederherzustellen. Weiterhin sind die meisten Programme zur forensischen Untersuchung in der Lage, gelöschte Dateien wieder sichtbar zu machen, wobei sich jedoch unterscheidet, welchen der genannten Ansätze die Software verfolgt⁷¹.

3.3 Schlupfspeicher

Neben der Analyse von Zeitstempeln und der Wiederherstellung gelöschter Dateien gilt die Analyse des sog. Schlupfspeichers (engl. slack space) eines Dateisystems als Quelle forensisch wertvoller Informationen⁷². Schlupfspeicher entsteht, wenn die Größe eines zu speichernden Elementes nicht dem Vielfachen einer Zuordnungseinheit entspricht⁷³. Im Rahmen eines FAT-Dateisystems kann Schlupfspeicher auf zwei Wegen entstehen, jeweils bedingt durch die festgelegte Größe einer Zuordnungseinheit.

Festplatten können nicht byteweise, sondern lediglich in Blöcken zu 512 Bytes (Sektoren) beschrieben werden⁷⁴. Somit entsteht zwischen dem Ende einer Datei und dem Ende eines Sektors ein Bereich, der von dem schreibenden Betriebssystem zu füllen ist. Dieser Bereich wird Schlupfspeicher genannt. Ältere Versionen von Microsoft DOS und Windows nutzten hierzu zufällig gewählte Daten aus dem Hauptspeicher des Computers; auf diese Weise werden teilweise sensible Daten unbeabsichtigt auf die Festplatte geschrieben und können im Rahmen einer forensischen Analyse ausgewertet werden. Aufgrund dieses Verhaltens wird der Schlupfspeicher zwischen dem Ende einer Datei und der Sektorgrenze als RAM-Slack bezeichnet⁷⁵. Neuere Versionen der Microsoft Betriebssysteme füllen diesen Bereich mit einem festgelegten Bitmuster; eine gewinnbringende Analyse des RAM-Slacks ist in diesem Fall nicht möglich⁷⁶. Die genauen Möglichkeiten einer Schlupfspeicheranalyse hängen jedoch von der Implementierung des FAT-Treibers des jeweils schreibenden Gerätes ab. Insbesondere bei mobilen Geräten, beispielsweise Digitalkameras, finden sich nach Erfahrung des Autors im Schlupfspeicher wertvolle Informationen. Dies können beispielsweise Metainformationen gelöschter Bilder einer Digitalkamera sein: auch wenn ein gesamtes Bild nicht wiedehergestellt werden kann, ist es möglich, mit Hilfe dieser Informationen die Verwendung der Kamera zu einem bestimmten Zeitpunkt nachzuweisen.

FAT verwendet, wie in 2.1.1 beschrieben, feste Zuordnungseinheiten, sog. Cluster, um Dateiinhalte zu speichern. Die Größe eines Clusters wird jeweils als ein Vielfaches

71 [Carrier 2005, S. 247], [Geschonneck 2004, S. 97]

72 [Steel 2006, S. 52], [Carrier 2005, S. 187–188], [Geschonneck 2004, S. 85]

73 [Carrier 2005, S. 187]

74 Vgl. hierzu Abschnitt 2.1.1 – *Überblick*

75 [Geschonneck 2004, S. 86]

76 [Carrier 2005, S. 187]

einer Sektorgröße angegeben und beträgt meist 2048 oder 4096 Bytes. Der Speicher zwischen dem letzten Sektor, der von einer Datei belegt wird und dem letzten Sektor des zum Speichern verwendeten Clusters stellt ebenfalls Schlupfspeicher dar. Der so entstehende sog. Drive-Slack wird im Gegensatz zum RAM-Slack nicht mit neuen Daten überschrieben. Auf diese Weise bleiben Sektorinhalte alter Dateien erhalten und können forensisch analysiert werden⁷⁷.

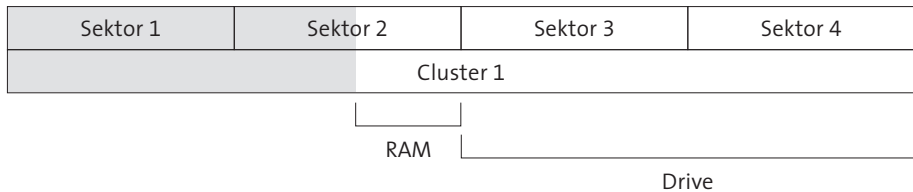


ABBILDUNG 6: RAM- UND DRIVE-SLACK EINES FAT-DATEISYSTEMS (CLUSTERGRÖSSE: 2048 BYTES)

Die oben stehende Abbildung illustriert die Unterschiede der Schlupfspeichertypen. Der grau hinterlegte Teil stellt hierbei die Datei dar, die den ersten Sektor sowie einen Teil des zweiten Sektors belegt. Der von dieser Datei nicht genutzte Bereich bis zum Ende des zweiten Sektors wird hierbei als RAM-Slack, der Speicherplatz in den Sektoren drei und vier als Drive-Slack bezeichnet.

3.4 Einschränkungen

Aufgrund der relativ einfachen Strukturen des FAT-Dateisystems⁷⁸ bestehen hinsichtlich der forensischen Analyse zwei gravierende Einschränkungen, die im Folgenden beschrieben werden. Die Kenntnis über diese Einschränkungen ist für eine forensische Untersuchung von fundamentaler Bedeutung, da andernfalls eine fehlerfreie Interpretation der erhaltenen Daten nicht möglich ist.

Abschnitt 3.1 erläutert die innerhalb eines Directory-Entries gespeicherten Zugriffszeiten einer Datei mit ihren jeweiligen Genauigkeiten. Diese schränken eine Analyse ein, da die Granularität des Zeitstempels in Abhängigkeit des Zugriffstyps stark variiert:⁷⁹

- Erstellungszeitpunkt: genau bis auf eine zehntel Sekunde
- Zuletzt geschrieben: genau bis auf 2 Sekunden
- Zuletzt zugegriffen: tagesgenau

⁷⁷ [Geschonneck 2004, S. 86], [Carrier 2005, S. 188]

⁷⁸ [Carrier 2005, S. 221]

⁷⁹ [Carrier 2005, S. 288], [Sammes 2007, S. 204]

Insbesondere die Einschränkung des Zeitpunktes des letzten Zugriffs führt dazu, dass keine minutengenaue Aussage über die Verwendung des Gerätes bzw. den (lesenden) Zugriff einer Datei getroffen werden kann. In Abhängigkeit der Fragestellung der Analyse kann dies jedoch erwünscht sein. In diesem Fall können nach Erfahrung des Autors Informationen zur letzten Nutzung eines Gerätes jedoch häufig mit Hilfe des Zeitstempels der letzten Änderung verschiedener Systemdateien hinreichend genau ermittelt werden. Der letzte Zeitpunkt eines lesenden Zugriffs dedizierter Dateien lässt sich jedoch auf keinem anderen Weg ermitteln.

Eine weitere Einschränkung ergibt sich aus der fehlenden Benutzer- und Rechtestruktur des FAT-Dateisystems. Bei Mehrbenutzersystemen wie beispielsweise Microsoft Windows kann somit nicht nachvollzogen werden, welcher Benutzer eines Computers die Datei erstellt oder geschrieben hat. Hierbei können die Windows-Ereignisse unterstützende Informationen enthalten. Dies hängt jedoch davon ab, wie diese konfiguriert sind (Zeitraum der Aufbewahrung, Granularität der Ereignisaufzeichnung)⁸⁰.

4 Fazit und Ausblick

FAT gilt, aufgrund weniger und einfacher Datenstrukturen, als eines der einfachsten Dateisysteme⁸¹. Im Laufe der Zeit haben sich, nach der ersten Version aus den 1970er Jahren, verschiedene Versionen entwickelt, wobei zum Erstellungszeitpunkt dieser Arbeit FAT32 den höchsten Verbreitungsgrad besitzt. Wegen der einfachen Struktur und der geringen Anzahl von Funktionen hat sich das FAT32-Dateisystem als Standarddateisystem für mobile Geräte wie Digitalkameras oder USB-Speichermedien durchgesetzt, während auf Festplatten neuerer Windows-Systeme dessen Nachfolger, das New Technology File System (NTFS) verwendet wird⁸².

Bei der forensischen Analyse eines FAT-Dateisystems sind primär Datum und Uhrzeit von Belang, um daraus Informationen über die Benutzung des Computers abzuleiten. Diese können, mit den in Abschnitten 3.1 sowie 3.4 beschriebenen Einschränkungen, aus einem Verzeichniseintrag gewonnen werden. Weiterhin werden die Nutzdaten einer Datei beim Löschvorgang nicht entfernt; gleiches gilt für einige Metadaten, die nicht entfernt werden, sondern deren Speicherplatz lediglich zur erneuten Verwendung freigegeben wird. Aus den genannten Gründen ist es bei FAT-Dateisystemen möglich, gelöschte Dateien wieder herzustellen um daraus forensisch relevante Daten zu extrahieren.

Aufgrund der in Kapitel 2 beschriebenen Strukturen und Zusammenhänge kann es bei der Verwendung des FAT-Dateisystems zu sog. Schlupfspeichern kommen. Dabei

80 Für weitere Informationen zu Ereignissen unter Microsoft Windows siehe auch [Steel 2006, S. 247 ff].

81 [Carrier 2005, S. 211]

82 [Steel 2006, S. 66–68]

handelt es sich um Bereiche innerhalb des Dateisystems, die zwar einer Datei zugeordnet sind, aber, aufgrund einer festgelegten Cluster- bzw. Sektorengröße, keinen Dateiinhalt mehr tragen. Die Analyse dieses Schlupfspeichers kann im Rahmen der IT-Forensik Aufschlüsse über gelöschte bzw. ältere Datenbestände erlauben.

Um weiterhin wachsenden Anforderungen wie größer werdenden Dateien und Datenträgern sowie einer durchgängig abgebildeten Rechtestruktur Rechnung zu tragen, hat Microsoft 2006 einen Nachfolger für FAT32 angekündigt. Dieser trägt den Namen exFAT und soll primär auf mobilen und eingebetteten Geräten implementiert werden, da deren Leistung zur Nutzung des NTFS-Dateisystem meist nicht ausreichend ist. Eine Dokumentation wie für vorherige Dateisystemversionen ist jedoch nicht frei verfügbar. Eine Implementierung ist nur bei entsprechender Lizenzierung durch Microsoft möglich – diese setzt jedoch die Unterzeichnung einer Vertraulichkeitsvereinbarung voraus. Daher kann davon ausgegangen werden, dass eine vollständige, wissenschaftliche Beschreibung von exFAT nicht möglich ist.

5 Quellen

- [Carrier 2005] Carrier, B.: File System Forensic Analysis, Addison-Wesley, Crawfordsville, 2005
- [Geschonnek 2004] Geschonnek, A.: Computer Forensik, dpunkt.verlag, Heidelberg 2004
- [Microsoft 2000] FAT: General Overview of On-Disk Format, Hardware Whitepaper, Microsoft 2000 – fatgen103.doc <http://www.microsoft.com/whdc/system/platform/firmware/fatgen.msp>
- [Steel 2006] Steel, C.: Windows Forensics. The Field Guide for Conducting Corporate Computer Investigations, Wiley Publishing, Indianapolis 2006
- [Sammes 2007] Sammes, T., Jenkinson, B.: Forensic Computing, 2nd Edition, Springer, London 2007
- [Russinovich 2005] Russinovich, M. E., Solomon, D.: Windows Internals. Microsoft Windows Server 2003, Windows XP and Windows 2000, 4th edition, Microsoft Press, Redmond, Washington 2005

Das

NTFS Dateisystem

von **Stefan Humml**

Abstract

Das von Microsoft entwickelte Dateisystem „New Technology File System“ (NTFS) hat aufgrund der Tatsache, dass es für alle aktuellen Microsoft Betriebssysteme das Standard-Dateisystem darstellt, einen hohen Verbreitungsgrad. IT-forensische Untersuchungen verlangen ein gutes Verständnis der eingesetzten Dateisysteme; dieses wird im Rahmen der vorliegenden Arbeit vermittelt. Darüber hinaus werden auf Basis der technischen Grundlagen IT-forensische Ansätze für das NTFS-Dateisystem erläutert. Dies umfasst die Lokationen verschiedener Zeitstempel genauso wie die Wiederherstellung gelöschter Dateien oder potenzielle Informationsgewinne aus dem Schlupfspeicher (Slack).

Inhaltsverzeichnis

| | |
|---|----|
| 1 Zielsetzung und Vorgehensweise | 47 |
| 1.1. Anmerkungen..... | 47 |
| 2 Einführung..... | 48 |
| 2.1 Funktionen des Dateisystems..... | 48 |
| 3. Strukturen und Konzepte..... | 50 |
| 3.1 Überblick | 50 |
| 3.2 MFT..... | 51 |
| 3.3 Fixups | 53 |
| 3.4 Metadaten-Dateien..... | 54 |
| 3.4.1 \$Boot..... | 55 |
| 3.4.2 \$MFTMirr..... | 56 |
| 3.4.3 Weitere Metadaten-Dateien | 57 |
| 3.5 Attribute..... | 58 |
| 3.5.1 Externer Attribut-Inhalt: Cluster-Runs | 60 |
| 3.5.2 Standard-Attribute | 61 |
| 3.5.2.1 \$STANDARD_INFORMATION und \$FILE_NAME..... | 62 |
| 3.5.2.2 \$DATA..... | 64 |
| 3.5.2.3 Weitere Attribute | 64 |
| 3.6 Indizes | 64 |
| 4 Analyse..... | 68 |
| 4.1 Datum und Uhrzeit | 68 |
| 4.2 Alternate Data Streams (ADS)..... | 70 |
| 4.3 Wiederherstellen gelöschter Dateien | 71 |
| 4.4 Weitere Analysemöglichkeiten..... | 73 |
| 5 Fazit und Ausblick..... | 74 |
| 6 Quellen | 76 |

Abbildungsverzeichnis

| | | |
|---------------|---|----|
| Abbildung 1: | Layout eines NTFS-Datenträgers | 5 |
| Abbildung 2: | Skizze MFT-Eintrag..... | 7 |
| Abbildung 3: | Exemplarische Datenstruktur mit und ohne Fixups | 8 |
| Abbildung 4: | Skizze MFT-Eintrag. Erweitert um Attribut-Köpfe..... | 12 |
| Abbildung 5: | Cluster-Runs an einem Beispiel | 14 |
| Abbildung 6: | Skizze eines Runlist-Eintrags | 15 |
| Abbildung 7: | Exemplarischer B-Baum | 19 |
| Abbildung 8: | Struktur eines NTFS-Index-Knotens | 20 |
| Abbildung 9: | Exemplarischer Aufbau eines MFT-Eintrags mit \$INDEX_ALLOCATION..... | 21 |
| Abbildung 10: | RAM- und Drive-Slack eines NTFS-Dateisystems (Clustergröße: 2048 Bytes)..... | 28 |

Tabellenverzeichnis

| | | |
|------------|---|----|
| Tabelle 1: | Struktur eines MFT-Eintrags (file record) | 8 |
| Tabelle 2: | Übersicht über Metadaten-Dateien | 9 |
| Tabelle 3: | Aufbau der \$Boot-Datei..... | 11 |
| Tabelle 4: | Standardkopf eines Attributs | 13 |
| Tabelle 5: | Erweiterter Kopf eines residenten Attributs | 13 |
| Tabelle 6: | Erweiterter Kopf eines nicht-residenten Attributs | 14 |
| Tabelle 7: | Standard MFT-Attribute | 16 |
| Tabelle 8: | Struktur des \$STANDARD_INFORMATION-Attributs | 17 |
| Tabelle 9: | Struktur des \$FILE_NAME-Attributs | 18 |

Abkürzungsverzeichnis

| | |
|------|--|
| ADS | Alternate Data Stream |
| EFS | Encrypting File System |
| FAT | File Allocation Table |
| HPFS | High Performance File System |
| JPEG | Joint Photographic Experts Group |
| LCN | Logical Cluster Number |
| LSN | Logfile Sequence Number |
| MP3 | MPEG-1 Audio Layer 3 |
| MPEG | Moving Picture Experts Group |
| NT | New Technology |
| NTFS | New Technology File System |
| USB | Universal Serial Bus |
| UTC | Universal Time Coordinated |
| VCN | Virtual Cluster Number |
| XP | Experience (gilt für Microsoft-Produkte) |

1 Zielsetzung und Vorgehensweise

Das NTFS¹ Dateisystem bildet seit der Einführung in Windows NT² die Grundlage der Datenspeicherung unter Windows-Betriebssystemen auf Servern. Mit Microsoft Windows XP³ wurde NTFS ebenfalls als Standard auf Heimcomputern etabliert⁴. Aufgrund des hohen Verbreitungsgrades der genannten Betriebssysteme findet NTFS eine starke, weltweite Verbreitung. Um eine exakte, vollständige und nachvollziehbare forensische Untersuchung dieser Datenträger durchführen zu können, ist es notwendig, die Strukturen und Konzepte sowie Analysemöglichkeiten des Dateisystems zu kennen.

Ziel dieser Arbeit ist die systematische Erläuterung des Dateisystems NTFS sowie die Beschreibung möglicher forensischer Analyseansätze und deren Einschränkungen. Hierzu stellen Kapitel 2 und 3 das Dateisystem vor. Dabei wird zunächst ein Überblick gegeben, nachfolgend werden einzelne elementare Strukturen des Dateisystems sowie deren Zusammenhänge beschrieben.

Kapitel 4 erläutert forensische Analysemöglichkeiten des Dateisystems sowie zu beachtende Einschränkungen. Die Arbeit schließt mit einem zusammenfassenden Fazit und einem Ausblick.

1.1 Anmerkungen

Zu NTFS existieren keine offiziellen, detaillierten Dokumentationen seitens Microsoft⁵. Die Angaben dieser Arbeit beziehen sich primär auf Untersuchungen von Brian Carrier⁶ und Informationen des Linux-NTFS-Projektes⁷ sowie Beobachtungen und Analysen von Sammes und Jenkinson⁸. Es sei betont, dass Microsoft jederzeit die Möglichkeit hat, das Dateisystem anzupassen bzw. zu verändern, ohne dies zu dokumentieren.

Innerhalb dieser Arbeit werden nicht alle Aspekte und Funktionen des NTFS-Dateisystems ausführlich erläutert, da dies nicht dem Rahmen der Arbeit entspräche und weiterhin zum Verständnis der Funktionsweise des Dateisystems nicht notwendig ist. Abschnitt 2.1 beschreibt jedoch in Kürze alle Möglichkeiten bzw. Optionen des Dateisystems. Für detaillierte Beschreibungen und Analysen der entsprechenden Funktionalitäten sei an dieser Stelle auf [Carrier 2005], [Russonovich 2009] und [Russon 05] verwiesen.

1 New Technology File System

2 New Technology

3 experience

4 [Sammes 2007, S. 216]

5 Siehe hierzu [Carrier 2005, S. 274]

6 [Carrier 2005]

7 [Russon 05]

8 [Sammes 2007]

Alle Zahlenangaben dieser Arbeit, insbesondere innerhalb von Tabellen, beziehen sich auf das Dezimalsystem. Hexadezimale Zahlenangaben werden durch vorangestelltes „0x“ kenntlich gemacht.

2 Einführung

Das *New Technology File System* (NTFS) wurde von Microsoft als Nachfolger von FAT⁹ entwickelt, um sowohl wachsenden Datenmengen als auch Anforderungen der Benutzer wie beispielsweise Rechtvergabe oder eine höhere Robustheit gegenüber Inkonsistenzen und Fehlern Rechnung zu tragen. Es ist mit Windows NT eingeführt und seitdem stetig weiterentwickelt worden. Seit Windows XP wird es auch für Heimnutzer als Standarddateisystem eingesetzt und ist aufgrund dessen sehr weit verbreitet.¹⁰

Im Folgenden wird ein Überblick über Dateisystemversion und korrespondierendem Betriebssystem gegeben¹¹:

- NTFS 1.x – Windows NT 3,1 und Windows NT 3,5
- NTFS 2.x – Windows NT 4,0
- NTFS 3,0 – Windows 2000
- NTFS 3,1 – Windows XP, Windows Vista, Windows Server 2003, Windows Server 2008 und Windows 7

Der nachfolgende Abschnitt beschreibt Funktionen des NTFS-Dateisystems. Jedoch werden nicht alle im Folgenden genannten Funktionen des Dateisystems im Rahmen dieser Arbeit detailliert beschrieben¹².

2.1 Funktionen des Dateisystems

Das NTFS-Dateisystem bietet eine Liste von Funktionen, die sein Vorgänger FAT nicht zur Verfügung stellte. Diese werden im Folgenden kurz skizziert.

NTFS ermöglicht es, Dateien komprimiert zu speichern und auf diese Weise Speicherplatz auf dem Datenträger zu sparen. Die dabei eingesetzten Kompressionsalgorithmen sind jedoch nicht dokumentiert, so dass eine umfassende Beschreibung

9 File Allocation Table

10 [Sammes 2007, S. 215], [Carrier 2005, S. 273–274], [Steel 2006, S. 75–76]

11 [Russon 05, S. 4]. Bei den hier genannten Versionen handelt es sich um die Versionsnummer, die auf der Festplatte hinterlegt wird. Andere Autoren, z.B. [Sammes 2007, S. 217] verwenden eine andere Nomenklatur, wobei Microsoft selbst die in dieser Arbeit gezeigten Versionsnummern verwendet. Siehe hierzu auch [Russinovich 2009, S. 937]

12 Siehe hierzu auch Abschnitt 1.1 – Anmerkungen

nicht möglich ist¹³. Weiterhin ermöglicht NTFS die *Verschlüsselung* von Dateien. Microsoft bezeichnet die Verschlüsselung bei NTFS als *Encrypting File System (EFS)*¹⁴. Dem Benutzer wird im Rahmen von EFS die Möglichkeit gegeben, einzelne Dateien und Verzeichnisse zu verschlüsseln; die Verschlüsselung der gesamten Festplatte erfolgt nicht mit Hilfe von EFS, sondern wird durch das Produkt Bitlocker realisiert¹⁵. Die Dateiverschlüsselung auf Basis von EFS wird ab Version 3.0 des Dateisystems unterstützt¹⁶.

Eine weitere Funktionalität des NTFS Dateisystems ist das sog. Journaling. Dabei werden Änderungen der Datenstrukturen des Dateisystems zunächst in ein Journal, vergleichbar mit einem Aufgabenbuch, geschrieben, bevor sie tatsächlich umgesetzt werden. Dies ermöglicht ein effizientes Reagieren auf Fehler, die beispielsweise bei einem Stromausfall auftreten können und erhöht somit die Zuverlässigkeit des Dateisystems¹⁷. Das Journal enthält zu diesem Zweck folgende Informationen¹⁸:

- Zeit und Datum der Änderung
- Grund der Änderung (z.B. „eine Datei wurde umbenannt“)
- Attribute der geänderten Datei
- Name der geänderten Datei
- Referenznummer der Datei
- Referenznummer des Verzeichnisses, in dem die Datei liegt
- Security-ID
- Update Sequence Nummer der Änderung

Mit Hilfe dieser Informationen kann im Falle eines Computerausfalls oder Fehlers des Dateisystems (z.B. Entfernen eines Datenträgers während eines Schreib- oder Änderungsvorgangs) ein konsistenter Zustand des Dateisystems wiederhergestellt werden¹⁹.

Sog. *Quotas* (dt. Quoten) stellen ein Mittel für Administratoren dar, um den Speicherbereich für einzelne Nutzer einzuschränken. Hierbei hat der Dateisystemtreiber bzw. das Betriebssystem die Aufgabe, die Datenmengen eines jeden Benutzers auf Basis seiner Benutzer-ID zu verwalten und bei Überschreitung eines durch den Administrator gesetzten Limits weitere Schreibvorgänge zu verweigern²⁰. Dies dient der Sicherung der Verfügbarkeit des Dateisystems; einzelne Benutzer sind nicht in der Lage, den gesamten verfügbaren Speicher zu allokalieren und somit Schreibvorgänge anderer Benutzer zu blockieren.

13 [Carrier 2005, S. 285 ff.]

14 [Carrier 2005, S. 287]

15 Weitere Informationen zu Bitlocker bei [Russinovich 2009, S. 677–684]

16 [Steel 2006, S. 88]

17 [Carrier 2005, S. 340]

18 [Russinovich 2009, S. 956–959]

19 [Russinovich 2009, S. 974]

20 [Russinovich 2009, S. 962 f.]

Weiterhin ist in NTFS eine umfangreiche *Rechteverwaltung* implementiert, die es ermöglicht, Benutzern auf Dateien und Ordnern verschiedene Rechte zu geben²¹. Diese Berechtigungskonzepte sind für die forensische Analyse, insbesondere bei Mehrbenutzersystemen relevant und werden im Rahmen der Analyse in Abschnitt 4.4 behandelt.

Sog. *Sparse-Files* (dt. sparsame Dateien) ermöglichen es, große Dateien anzulegen, die eine Reihe von Null-Bytes enthalten. Dabei werden die Null-Bytes jedoch nicht physikalisch allokiert, solange sie nicht anderen Daten beschrieben werden. Auf diese Weise kann auf dem Datenträger Platz gespart werden.²²

Die in diesem Abschnitt beschriebenen Funktionalitäten stellen, mit Ausnahme der Rechteverwaltung, optionale Funktionalitäten des Dateisystems dar. Ihr Verständnis ist für einen grundlegenden Überblick über NTFS nicht notwendig. Im Gegensatz dazu beschreibt das folgende Kapitel elementare Bausteine des Dateisystems sowie deren Zusammenhänge, die die Funktionsweise von NTFS ausmachen und deren Verständnis für eine profunde Analyse aus Sicht des Autors zwingend notwendig ist.

3 Strukturen und Konzepte

Im Folgenden werden die einzelnen Konzepte und Datenstrukturen des NTFS-Dateisystems detailliert beschrieben, die notwendig sind, um Aufbau und Funktionsweise des Dateisystems zu verstehen. Abschnitt 3.1 gibt zu diesem Zweck zunächst eine Übersicht über das Layout des Dateisystems sowie grundlegende Definitionen. Die nachfolgenden Abschnitte beschreiben einzelne Elemente des Dateisystems und deren Zusammenwirken.

3.1 Überblick

In NTFS wird jegliche Information in einer Datei gespeichert. Insbesondere sind Strukturen, die das Dateisystem beschreiben, sog. Metadaten, ebenfalls in Dateien gespeichert. Aus diesem Grund wird der gesamte Datenträger, der mit Hilfe von NTFS formatiert wurde, als Datenbereich angesehen.²³

Das Layout eines NTFS-Datenträgers kann folgendermaßen skizziert werden²⁴.

21 [Russinovich 2009, S. 963–965].

22 [Carrier 2005, S. 191 und S. 284]. Microsoft spricht in Zusammenhang mit Sparse-Files auch von Kompression [Russinovich 2009, S. 927]

23 [Carrier 2005, S. 273 f.]

24 Zwar handelt es sich insgesamt um den Datenbereich, dieser kann jedoch weiter aufgeteilt werden. Das hier dargestellte Layout entspricht dem Standard-Layout von NTFS. Das Dateisystem ist jedoch flexibel genug, um alle Dateien (mit Ausnahme von \$Boot) verschieben zu können. [Russon 05, S. 5]

| | | | | |
|------|-----------|------|-----------|------|
| Boot | Metadaten | frei | Metadaten | frei |
|------|-----------|------|-----------|------|

ABBILDUNG 1: LAYOUT EINES NTFS-DATENTRÄGERS²⁵

NTFS verwendet *Cluster*, um Informationen (und Daten) zu speichern und zu adressieren. Es handelt sich hierbei um Blöcke zusammenhängender Sektoren auf einem Datenträger. Die Anzahl Sektoren pro Cluster wird durch einen Eintrag in der \$Boot-Datei festgelegt²⁶. Es muss sich jedoch um eine Potenz von zwei handeln²⁷. Der erste Cluster eines NTFS-formatierten Dateisystems beginnt mit dem ersten Sektor des Volumes und trägt die Nummer Null. Somit ist eine direkte Umrechnung zwischen Cluster und Sektor möglich²⁸:

$$\text{Sektor} = \text{Cluster} * \text{Sektoren_pro_cluster}$$

Weiterhin speichert NTFS alle Datentypen im Little-Endian-Format²⁹.

Generell werden Datumsangaben bzw. Uhrzeiten in NTFS als 64-Bit Wert gespeichert. Beispielsweise enthält die Information zu einer Datei vier verschiedene Zeitstempel, die auf die anschließend erläuterte Weise gespeichert werden³⁰:

- Zeitpunkt des Erstellens
- Zeitpunkt der letzten Änderung
- Zeitpunkt der letzten Modifikation des MFT-Eintrags³¹
- Zeitpunkt des letzten Zugriffs

Die entsprechenden Einträge innerhalb des Dateisystems beziehen sich auf die Anzahl der seit dem 01.01.1601 UTC³² verstrichenen Einhundert-Nanosekunden-Anteile.³³

3.2 MFT

Die wichtigste Datei eines NTFS-Dateisystems ist die sog. *Master File Table* (MFT). Sie speichert für jedes Verzeichnis und jede Datei vitale Informationen. Jeder Eintrag der MFT³⁴ ist genau 1024 Bytes groß. Jedoch ist lediglich der Inhalt der ersten 42 Bytes,

²⁵ Angelehnt an [Russon 05, S. 4]

²⁶ Hierbei handelt es sich um einen Dateieintrag für den Boot-Sector. Vgl. Abschnitt 3.4.1 – \$Boot

²⁷ Siehe hierzu auch [Russonovitch 2009, S. 937 f]

²⁸ [Carrier 2005, S. 311]

²⁹ [Russon 05, S. 1]

³⁰ Siehe hierzu ebenfalls Abschnitt 4.1 – Datum und Uhrzeit

³¹ Siehe hierzu das folgende Kapitel 3.2.

³² Universal Time Coordinated, dt. koordinierte Weltzeit

³³ [Carrier 2005, S. 317]

³⁴ Microsoft bezeichnet die Einträge der MFT als „file record“. [Carrier 2005, S. 275]

der sog. *Header* (dt. Kopf) definiert. Alle nachfolgenden Informationen werden durch Attribute³⁵ dargestellt, deren Verwendung und Vorhandensein für jede Datei unterschiedlich sein kann. Auf diese Weise wird eine hohe Flexibilität erreicht.³⁶

Folgende Abbildung skizziert einen MFT-Eintrag.

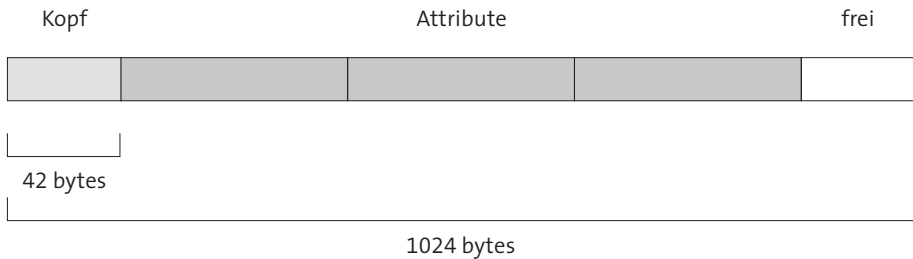


ABBILDUNG 2: SKIZZE MFT-EINTRAG³⁷

Wie in Abschnitt 3.1 beschrieben, sind alle Informationen und Daten in NTFS als Dateien organisiert. So auch die Master File Table. Ihr Name lautet *\$MFT*. Eine Besonderheit besteht darin, dass die *\$MFT*-Datei einen Eintrag für sich selbst enthält; dieser muss zunächst analysiert werden, um auf die Inhalte des Dateisystems zugreifen zu können bzw. erste Informationen über die *\$MFT*-Datei selbst zu erhalten (z.B. die Größe der *\$MFT*). Die Lage der *\$MFT*-Datei wird durch den Boot-Bereich bestimmt.³⁸

Die Größe der MFT-Datei wird von Windows bei der Formatierung eines Datenträgers möglichst klein gewählt. Da es sich um eine herkömmliche Datei handelt, kann die Größe bei Bedarf, z.B. bei wachsender Menge gespeicherter Dateien, leicht angepasst werden. Um hierbei jedoch eine *Fragmentierung* zu vermeiden, wird ein Bereich des Datenträgers für die MFT reserviert. Dieser Bereich wird als *MFT-Zone* bezeichnet.³⁹

Die Adressierung der MFT-Einträge erfolgt über einen 48-Bit-Wert und beginnt bei 0. Jeder Eintrag der Master-File-Table ist genau 1024 Bytes groß. Somit lässt sich ein Eintrag direkt adressieren.⁴⁰

Die folgende Tabelle stellt die genauen Informationen für einen MFT-Eintrag (*file record*) dar:

35 Attribute können z.B. Dateinamen, Verzeichnisinformationen oder Dateiinhalte sein. Siehe hierzu Abschnitt 3.5 – *Attribute*

36 [Carrier 2005, S. 274]

37 Angelehnt an [Carrier 2005, S. 275].

38 [Carrier 2005, S. 275]

39 I.d.R. allokiert ein Microsoft Betriebssystem 12,5% der Volumegröße für die MFT. [Russon 05, S. 5], [Carrier 2005, S. 313]

40 [Carrier 2005, S. 353]

| Offset | Größe | Beschreibung |
|--------|-------|--|
| 0 | 4 | Signatur (z.B. „FILE“, „BAAD“) |
| 4 | 2 | Offset zu Fixup-Array ⁴¹ |
| 6 | 2 | Anzahl Einträge des Fixup-Arrays |
| 8 | 8 | \$LogFile Sequenznummer (LSN) |
| 16 | 2 | Sequenz-Nummer |
| 18 | 2 | Link-Zähler |
| 20 | 2 | Offset zum ersten Attribut |
| 22 | 2 | Flags |
| 24 | 4 | Benutzter Platz des MFT-Eintrags |
| 28 | 4 | Allokierter Platz des MFT-Eintrags |
| 32 | 8 | Referenz zum Basis-Eintrag |
| 40 | 2 | Nächste Attribut-Kennung |
| 42 | 982 | Attribute und Fixups (in Abhängigkeit von Datei- bzw. Verzeichnistyp) |

TABELLE 1: STRUKTUR EINES MFT-EINTRAGS (FILE RECORD)⁴²

3.3 Fixups

Wie in Abschnitt 3.1 beschrieben, werden Cluster als ein Vielfaches eines Sektors dargestellt. Um innerhalb eines Clusters feststellen zu können, ob einer der Sektoren defekt ist, verwendet NTFS sog. Fixup-Werte für jeden Sektor⁴³. Dies bedeutet, dass in jedem Sektor, der einem allokierten Cluster zugeordnet ist, die letzten zwei Bytes durch einen definierten Wert ersetzt werden⁴⁴. Wird beim Lesen eines Sektors festgestellt, dass sich die erwarteten Informationen nicht an dessen Ende befinden, gilt der Sektor als fehlerhaft⁴⁵. Dabei gehen jedoch Informationen der Datei verloren. Diese werden in dem Header gespeichert, der einem durch Fixups gesicherteren Bereich vorangeht. Folgende Abbildung verdeutlicht diesen Zusammenhang:

41 Zu Fixups siehe nachfolgenden Abschnitt

42 Vgl. [Carrier 2005, S. 353] und [Russon 05, S. 89]

43 Fixups werden nicht für Dateiinhalte, sondern für Metainformationen verwendet. [Carrier 2005, S. 325]

44 Es wird die sog. Datei-Signatur verwendet. Siehe hierzu [Carrier 2005, S. 276 f.]

45 Dies führt beispielsweise zu einer Re-Allokation der Metainformationen.

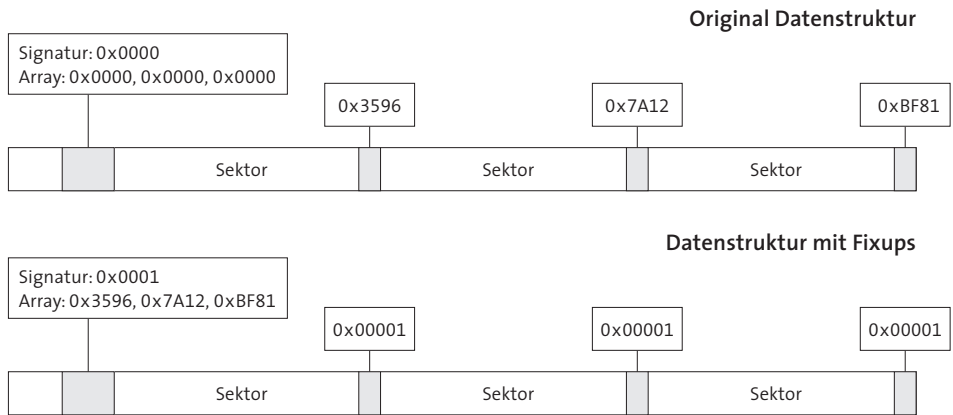


ABBILDUNG 3: EXEMPLARISCHE DATENSTRUKTUR MIT UND OHNE FIXUPS⁴⁶

Liest der Dateisystemtreiber die in der Abbildung skizzierten Sektoren und stellt währenddessen fest, dass nicht der erwartete Wert (0x0001 in diesem Beispiel) am Ende eines Sektors zu finden ist, so wird dieser Sektor als defekt behandelt. Die bei der Erstellung der Fixups überschriebenen Daten, jeweils zwei Bytes pro Sektor, werden, wie in der Abbildung angedeutet, in den Header einer mit Fixups versehenen Datei geschrieben und vom Dateisystemtreiber bei normalen Lesevorgängen an der entsprechenden Stelle eingefügt.

3.4 Metadaten-Dateien

Da in NTFS alle Informationen in Dateien organisiert sind, werden Dateien auch verwendet, um essentielle *Metadaten* für das Dateisystem zu speichern. Diese Metadaten-Dateien werden ebenfalls in der \$MFT-Datei geführt. Zu diesem Zweck werden die ersten 16 Einträge der MFT frei gehalten bzw. mit definierten Dateien belegt. Folgende Tabelle gibt einen Überblick über die Metadaten-Dateien, ihren MFT-Eintrag und Verwendungszweck⁴⁷:

| MFT-Nr. | Dateiname | Beschreibung |
|---------|-----------|--|
| 0 | \$MFT | Eintrag für die MFT-Datei selbst |
| 1 | \$MFTMirr | Eintrag für die Sicherheitskopie der MFT |
| 2 | \$LogFile | Enthält das Journal des Dateisystems |

46 [Carrier 2005, S. 352]

47 Die Dateinamen der Metadaten-Dateien beginnen jeweils mit einem „\$“-Zeichen. Der darauf folgende Buchstabe wird als Großbuchstabe dargestellt. Vgl. [Carrier 2005, S. 278]

| | | |
|-------|-------------|---|
| 3 | \$Volume | Informationen zum Volume (Datenträger) wie z.B. Version |
| 4 | \$AttrDef | Informationen zu Attributen ⁴⁸ |
| 5 | . | Eintrag für das root- (Wurzel-) Verzeichnis des Datenträgers |
| 6 | \$Bitmap | Enthält Allokationsinformationen für jeden Cluster |
| 7 | \$Boot | Enthält den Boot-Sektor und Boot-Code |
| 8 | \$BadClus | Eine Liste nicht verwendbarer, defekter Sektoren |
| 9 | \$Secure | Informationen für die Datei-Sicherheit wie z.B. Zugriffsschutz |
| 10 | \$Upcase | Eine Liste mit Großbuchstaben der Unicode-Zeichen |
| 11 | \$Extend | Ein Verzeichnis, das Dateien für erweiterte Funktionali- täten des Dateisystems bereithält (z.B. Quotas) |
| 12–15 | [Unbenutzt] | – |

TABELLE 2: ÜBERSICHT ÜBER METADATEN-DATEIEN⁴⁹

Zwar handelt es sich bei den Metadaten-Dateien im Sinne von NTFS um normale Einträge, jedoch sind die Dateien für den Endbenutzer nicht sichtbar.

Die folgenden Abschnitte 3.4.1–3.4.3 beschreiben ausgewählte, für das Grundverständnis des Dateisystems relevante Metadaten-Dateien. Für weitere Informationen zu Metadaten-Dateien sei auf [Carrier 2005] und [Russon 05] verwiesen.

3.4.1 \$Boot

Die Datei *\$Boot* enthält Informationen, die beim Start des Betriebssystems (dem sog. *Booten*) benötigt werden. Ihre Lage auf dem Datenträger ist, im Gegensatz zu allen anderen Metadaten-Dateien, festgelegt und beginnt mit dem ersten Sektor des Volumens⁵⁰. Wie Tabelle 2 zu entnehmen ist, wird die *\$Boot*-Datei als MFT-Eintrag Nummer sieben geführt.

In der *\$Boot*-Datei werden Informationen gespeichert, die für den Zugriff auf das Dateisystem essentiell sind. Insbesondere Informationen wie Bytes pro Sektor und Sektoren pro Cluster (Offsets 11 und 13) dienen zur initialen Berechnung der Dateisystem-Geometrie. Die Struktur der *\$Boot*-Datei ist der folgenden Tabelle zu entnehmen. Die wichtigsten Werte befinden sich an Offset 13, 48 und 64: Anzahl Sektoren pro Cluster, Verweis zur MFT-Datei, Größe eines MFT-Eintrags⁵¹.

48 Vgl. hierzu *Abschnitt 3.5*

49 [Carrier 2005, S. 278], [Russon 05, S. 35] und [Russinovich 2009, S. 939]

50 [Carrier 2005, S. 275]

51 Trotz der variablen Größe in der *\$Boot*-Datei ist die Größe eines MFT-Eintrags nach [Carrier 2005, S. 276] immer 1024 Bytes.

| Offset | Größe | Beschreibung |
|--------|-------|---|
| 0 | 3 | Sprung-Anweisung (Assembler-Instruktion), um den Boot-Code zu erreichen |
| 3 | 8 | System-Kennung: „NTFS“ |
| 11 | 2 | Bytes pro Sektor |
| 13 | 1 | Sektoren pro Cluster |
| 14 | 2 | Anzahl reservierter Sektoren (Null für NTFS) |
| 16 | 4 | Nicht verwendet |
| 21 | 1 | Medientyp. 0xF8 für Standard-Volumes (z.B. Festplatte), 0xF0 für entfernbare Datenträger wie beispielsweise USB-Sticks. |
| 22 | 2 | Nicht verwendet; muss null sein |
| 24 | 8 | Nicht verwendet |
| 32 | 4 | Nicht verwendet; muss null sein |
| 36 | 4 | Nicht verwendet |
| 40 | 8 | Anzahl Sektoren im gesamten Volume |
| 48 | 8 | Verweis zum Startcluster der MFT |
| 56 | 8 | Verweis zum \$DATA-Attribut der Datei \$MFTMirr (Cluster) |
| 64 | 1 | Größe eines MFT-Eintrags (file record) |
| 65 | 3 | Nicht verwendet |
| 68 | 1 | Größe des Index-Records |
| 69 | 3 | Nicht verwendet |
| 72 | 8 | Seriennummer |
| 80 | 4 | Nicht verwendet |
| 84 | 426 | Boot-Code |
| 510 | 2 | Signatur (0xAA55) |

TABELLE 3: AUFBAU DER \$BOOT-DATEI⁵²

3.4.2 \$MFTMirr

Da die *MFT* für das Dateisystem vitale Informationen enthält, existiert eine Kopie deren wichtigster Inhalte: *\$MFTMirr*. Diese wird ebenfalls in der *MFT* referenziert und findet sich in Eintrag Nummer eins. Um die Wahrscheinlichkeit eines gemeinsamen Ausfalls zu minimieren, wird die *\$MFTMirr*-Datei i.d.R. in die Mitte des physikalischen Datenträgers gelegt.⁵³

Sollte die *\$MFT*-Datei beschädigt sein, muss das Betriebssystem dennoch in der Lage sein, auf die Kopie bzw. deren Inhalt zuzugreifen. Dies kann mit Hilfe des *Boot-Sec-*

⁵² [Carrier 2005, S. 379 f]

⁵³ Jedoch ist die genaue Lage der Datei *\$MFTMirr* von der Version des Betriebssystems abhängig, das den Datenträger formatiert. Vgl. [Carrier 2005, S. 313 f]. Andere Autoren, z.B. [Sammes 2007, S. 218] und [Russinovich 2009, S. 939] sprechen allgemein von „der Mitte des Datenträgers“.

tors (*\$Boot*) geschehen, da hier die Adresse des *\$DATA-Attributs*⁵⁴ der *\$MFTMirr*-Datei direkt angegeben wird⁵⁵.

3.4.3 Weitere Metadaten-Dateien

Die Datei *\$Volume* findet sich in MFT-Eintrag drei und enthält *Versions-* und *Namensinformationen* über den Datenträger und das NTFS-Dateisystem. Diese Informationen werden mit Hilfe definierter Attribute gespeichert. Diese Attribute sollten deshalb ausschließlich in der *\$Volume*-Datei auftreten⁵⁶.

Die Datei *\$Bitmap* enthält für jeden Cluster des NTFS-Dateisystems ein Bit; dieses ist gesetzt, wenn der Cluster belegt ist. Findet sich in dem zu einem Cluster korrespondierenden Bit eine Null, so ist der Cluster frei und kann verwendet werden. Die *\$Bitmap*-Datei ermöglicht es somit dem Betriebssystem, schnell und effizient freie Bereiche im Dateisystem zu finden.⁵⁷

Um eine Liste beschädigter Cluster zu pflegen verwendet NTFS die Datei *\$BadClus*. Sie enthält ein *\$DATA*-Attribut, dem Cluster zugewiesen werden, wenn diese als defekt gelten.⁵⁸

Der sog. *Dot-Eintrag* bzw. *Root-Eintrag* (dt. Wurzeleintrag oder Wurzelverzeichnis) der MFT beschreibt das Basisverzeichnis des Dateisystems und findet sich generell in MFT-Eintrag Nummer fünf. Von diesem Verzeichnis gehen alle weiteren Einträge über Verzeichnisse und Dateien aus.⁵⁹

Die Datei *\$AttrDef* trägt Informationen wie Namen und Typnummern zu den Attributen, die MFT-Einträgen zugewiesen werden können⁶⁰. Zwar existieren jeweils *Standardwerte* für die jeweiligen Attribute, jedoch bietet NTFS mit Hilfe der *\$AttrDef*-Datei die Flexibilität, diese anpassen zu können.⁶¹

Es existieren eine Reihe weiterer Metadaten-Dateien; diese werden verwendet, um erweiterte Funktionalitäten des Dateisystems⁶² darzustellen. Deren Kenntnis ist zum Grundverständnis der NTFS-Funktionen jedoch nicht notwendig⁶³.

54 Das *\$DATA*-Attribut trägt wesentliche Information der *\$MFTMirr*-Datei. Zu den Datei-Attributen siehe *Abschnitt 3.5*

55 Vgl. *Tabelle 3, Offset 56*

56 [Carrier 2005, S. 305]

57 [Carrier 2005, S. 311 f.]

58 [Carrier 2005, S. 312]

59 [Carrier 2005, S. 334]

60 Zu Attributen siehe *Abschnitt 3.5 – Attribute*

61 [Carrier 2005, S. 305 f.]

62 Für eine Liste der erweiterten NTFS-Funktionalitäten siehe *Abschnitt 2.1 – Funktionen des Dateisystems*

63 Für eine Liste dieser Dateien siehe [Rusinovich 2009, S. 939 f.]

3.5 Attribute

Wie in Abschnitt 3.2 besprochen ist der Inhalt eines *MFT-Eintrags*, mit Ausnahme des Headers, nicht fest definiert. Je nach Verwendungszweck des Eintrags⁶⁴ werden Attribute verwendet, um notwendige Informationen zu verankern. Jedes der möglichen Attribute beinhaltet selbst einen *Header*, wobei dieser sich von Attribut zu Attribut unterscheidet. Angelehnt an Abbildung 2 kann diese somit detaillierter dargestellt werden:

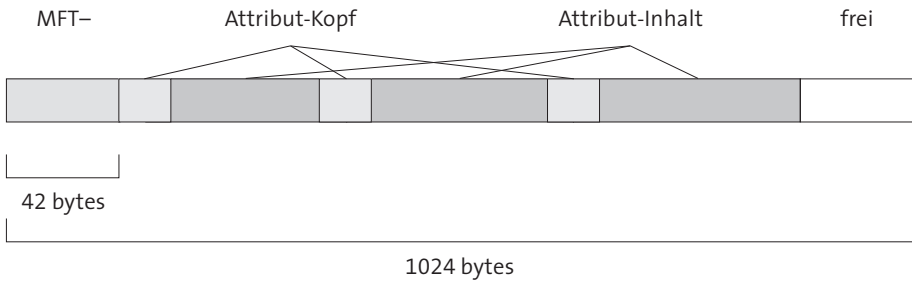


ABBILDUNG 4: SKIZZE MFT-EINTRAG. ERWEITERT UM ATTRIBUT-KÖPFE⁶⁵

Der *Attributkopf* (engl. *attribute header*) beschreibt durch eine Kennung, welches Attribut folgt. Da NTFS die Möglichkeit bietet, mehrere Attribute des gleichen Typs einem MFT-Eintrag zuzuordnen, trägt der Attributkopf weiterhin einen eindeutigen Identifikationswert (Offset 14). Die nachfolgende Tabelle beschreibt den Teil des Attributkopfes, der allen Attributen gemein ist:

| Offset | Größe | Beschreibung |
|--------|-------|--|
| 0 | 4 | Kennung (engl. attribute type identifier) |
| 4 | 4 | Länge des Attributs |
| 8 | 1 | Flag: liegen die Informationen innerhalb oder außerhalb des MFT-Eintrages? ⁶⁶ |
| 9 | 1 | Länge des Attributnamens |
| 10 | 2 | Offset zum Namen des Attributs (relativ zu Beginn des Attributs) |
| 12 | 2 | Flags |
| 14 | 2 | Eindeutige Kennung (attribute identifier) |

TABELLE 4: STANDARDKOPF EINES ATTRIBUTS⁶⁷

⁶⁴ Verwendungszweck kann z.B. ein Verzeichnis, eine Datei oder eine Metadatendatei sein.

⁶⁵ Angelehnt an [Carrier 2005, S. 279]

⁶⁶ Siehe nachfolgende Erläuterung *residenter bzw. nicht-residenter Attribute*.

⁶⁷ [Carrier 2005, S. 356]

Um den Inhalt eines Attributs, der nach dem Header folgt, zu speichern bieten sich zwei Möglichkeiten:

1. Der Inhalt kann direkt innerhalb des MFT-Eintrags gespeichert werden. Wird diese Möglichkeit gewählt, spricht man von *residenten* Attributen (engl. *resident attributes*).
2. Sollte der Platz innerhalb der MFT nicht ausreichen⁶⁸ kann der Inhalt eines Attributs in einem externen Cluster gespeichert werden. Somit entsteht ein nicht-residentes Attribut (engl. *non-resident attribute*).⁶⁹

Der dem Standardkopf eines Attributs nachfolgende Attributteil unterscheidet sich zwischen residenten und nicht-residenten Dateien:

| Offset | Größe | Beschreibung |
|--------|-------|---|
| 0 | 16 | [siehe Tabelle 4: Standardkopf eines Attributs] |
| 16 | 4 | Größe des Attribut-Inhaltes |
| 20 | 2 | Offset (Cluster) zum Inhalt |

TABELLE 5: ERWEITERTER KOPF EINES RESIDENTEN ATTRIBUTS⁷⁰

| Offset | Größe | Beschreibung |
|--------|-------|--|
| 0 | 16 | [siehe Tabelle 4: Standardkopf eines Attributs] |
| 16 | 8 | Start-VCN ⁷¹ der Runlist ⁷² |
| 24 | 8 | End-VCN der Runlist |
| 32 | 2 | Offset zu Runlist (relativ zum Anfang des Attributs) |
| 34 | 2 | Größe einer Kompressionseinheit |
| 36 | 4 | Nicht verwendet |
| 40 | 8 | Allokierter Platz für Attributinhalt |
| 48 | 8 | Tatsächliche Größe des Attributinhalt |
| 56 | 8 | Initialisierte Größe des Attributinhalt |

TABELLE 6: ERWEITERTER KOPF EINES NICHT-RESIDENTEN ATTRIBUTS⁷³

68 i.d.R. bietet ein MFT-Eintrag 1024 Bytes Platz. Vgl. *Abschnitt 3.2*.

69 Ein Attribut wird ab etwa 700 Bytes Größe extern geführt. [Carrier 2005, S. 283]

70 [Carrier 2005, S. 356]

71 Virtual Cluster Number

72 Siehe hierzu *Abschnitt 3.5.1 – Externer Attribut-Inhalt: Cluster-Runs*

73 [Carrier 2005, S. 357]

Der folgende Abschnitt beschreibt, wie nicht-residente Attributinhalt mit Hilfe sog. Cluster-Runs gespeichert werden.

3.5.1 Externer Attribut-Inhalt: Cluster-Runs

Nicht-residente Dateiattribute werden in sog. *Cluster-Runs* gespeichert. Dabei handelt es sich um Listen zusammenhängender Cluster, die gemeinsam den Inhalt eines Attributs darstellen.

Beispielsweise kann der Inhalt einer Datei auf folgende Cluster aufgeteilt werden: 48, 49, 50, 51, 52; 80, 81 und 56, 57, 58, 59. Daraus leiten sich Cluster-Runs ab, die wie folgt skizziert werden:

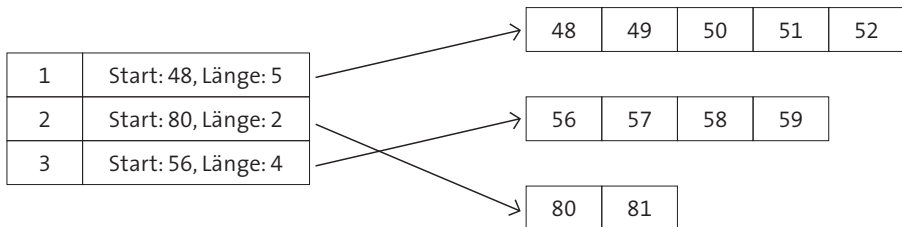


ABBILDUNG 5: CLUSTER-RUNS AN EINEM BEISPIEL⁷⁴

NTFS verwendet zur Adressierung der Cluster zwei verschiedene Methoden. Zum einen kann ein Cluster relativ zum Anfang des Inhalts eines Attributs adressiert werden. In diesem Fall spricht man von einer sog. *virtuellen Clusternummer* (engl. *virtual cluster number – VCN*). Wird ein Cluster jedoch relativ zum Anfang des Dateisystems adressiert handelt es sich um *logische Clusternummern* (engl. *logical cluster number – LCN*).⁷⁵

Die in Abbildung 5 genannten Clusternummern verwenden eine LCN-Adressierung. NTFS setzt eine Abbildung zwischen logischen und virtuellen Clusternummern ein. Die Zählung der virtuellen Clusternummern beginnt mit dem Inhalt des Attributs bei Null; somit kann im obigen Beispiel die Abbildung der logischen auf virtuelle Clusternummern erfolgen. Cluster 48 entspräche die virtuelle Cluster Nummer 0, Cluster 49 Nummer 1, Cluster 80 Nummer 5, Cluster 81 Nummer 6 usw.

Die Position der Cluster-Runlist wird innerhalb des Attributkopfes angegeben⁷⁶. Folgende Abbildung zeigt den Aufbau eines Runlist-Eintrags:

⁷⁴ [Carrier 2005, S. 281]

⁷⁵ [Russinovich 2009, S. 937 f.] und [Carrier 2005, S. 281]

⁷⁶ Siehe *Tabelle 6: Erweiterter Kopf eines nicht-residenten Attributs, Offset 32*

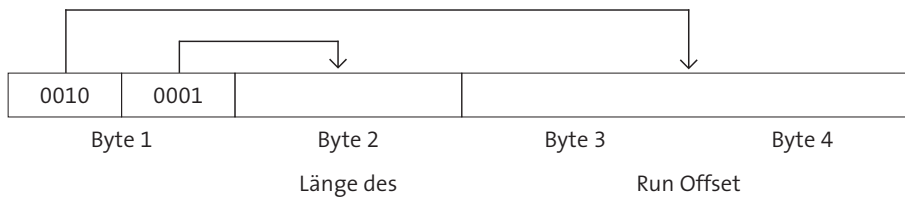


ABBILDUNG 6: SKIZZE EINES RUNLIST-EINTRAGS⁷⁷

Jeder Eintrag einer *Runlist* beginnt mit einem Byte, das die Größe der nachfolgenden Informationen definiert. Dazu wird das Byte in höher- und niederwertiges *Nibble* aufgeteilt. Das niederwertige *Nibble* beschreibt die Länge des Feldes, welches die Längeninformationen des Runs enthält. Das höherwertige *Nibble* hingegen legt die Länge des Feldes fest, das den *Offset* des Runs beschreibt. Die Angaben beziehen sich jeweils auf Cluster.⁷⁸

Das Offset eines *Runs* wird als *vorzeichenbehafteter Wert*⁷⁹ relativ zum vorhergehenden Offset angegeben. Da jedoch der erste Eintrag einer *Runlist* keinen vorhergehenden Eintrag besitzt, wird das entsprechende Offset relativ zum Anfang des Dateisystems angegeben.

Auf Basis dieser Informationen können sämtliche Cluster einer Datei adressiert und ausgelesen werden, auch wenn diese nicht zusammenhängend gespeichert sind. Weiterhin ist die Struktur der Metainformationen zu den Cluster-Runs so gewählt, dass möglichst wenig Speicherbedarf entsteht⁸⁰.

3.5.2 Standard-Attribute

Nachdem im vorigen Abschnitt dargelegt wurde, welchen Zweck Attribute haben, wie diese mit der MFT in Zusammenhang stehen und wie externe (nicht-residente) Attribute gespeichert werden, gibt dieser Abschnitt eine Übersicht über die gängigsten Datentypen des NTFS-Dateisystems⁸¹. Die folgende Tabelle listet alle in NTFS möglichen Attribute:

77 [Carrier 2005, S. 358]

78 [Carrier 2005, S. 358]

79 Es kann demzufolge auch negativ sein.

80 Vgl. *Abbildung 6: Skizze eines Runlist-Eintrags*

81 Generell werden NTFS-Attributnamen mit vorangehendem „\$“-Symbol geschrieben. Weiterhin bestehen ihre Namen aus Großbuchstabe; z.B. \$FILE_NAME. Dies dient der Unterscheidung zwischen Metadaten-Dateien, die auch Buchstaben in Kleinschreibweise verwenden. Vgl. [Carrier 2005, S. 281]

| ID-Nr. | Name | Beschreibung |
|--------|-------------------------|--|
| 16 | \$STANDARD_INFORMATION | Standardinformationen. Flags, Zeiten, Sicherheits-ID |
| 32 | \$ATTRIBUTE_LIST | Liste, die angibt, wo weitere Attribute gefunden werden können |
| 48 | \$FILE_NAME | Dateiname (Unicode), Zeiten |
| 64 | \$VOLUME_VERSION | Volume-Informationen (nur bis Windows NT) |
| 64 | \$OBJECT_ID | Eindeutiger Wert zur Identifikation der Datei |
| 80 | \$SECURITY_DESCRIPTOR | Sicherheits-Daten |
| 96 | \$VOLUME_NAME | Name des Volumes |
| 112 | \$VOLUME_INFORMATION | Dateisystemversion und Flags |
| 128 | \$DATA | Inhalt einer Datei |
| 144 | \$INDEX_ROOT | Wurzelelement eines Index-Baums ⁸² |
| 160 | \$INDEX_ALLOCATION | Elemente eines Index-Baums |
| 176 | \$BITMAP | Bitmap für Indizes |
| 192 | \$SYMBOLIC_LINK | Softlink-Informationen (nur bis Windows NT) |
| 192 | \$REPARSE_POINT | Informationen zu einem Reparse-Punkt. Dabei handelt es sich bei NTFS-Version >3.0 um Softlinks |
| 208 | \$EA_INFORMATION | Für OS/2 Abwärtskompatibilität (HPFS) ⁸³ |
| 224 | \$EA | Für OS/2 Abwärtskompatibilität (HPFS) |
| 256 | \$LOGGED_UTILITY_STREAM | Schlüssel und weitere Informationen bei verschlüsselten NTFS-Systemen |

TABELLE 7: STANDARD MFT-ATTRIBUTE⁸⁴

Nahezu jeder MFT-Eintrag besitzt die Attribute *\$STANDARD_INFORMATION* und *\$FILE_NAME*⁸⁵. Diese werden, neben anderen wichtigen Attributen, im Folgenden vorgestellt.

3.5.2.1 \$STANDARD_INFORMATION und \$FILE_NAME

Das Attribut *\$STANDARD_INFORMATION* ist generell resident und existiert für jede Datei und jedes Verzeichnis eines NTFS-Dateisystems⁸⁶. Folgende Tabelle stellt die Inhalte des Attributs dar:

⁸² Siehe hierzu Abschnitt 3.6. – *Indizes*

⁸³ High Performance File System. Hierbei handelt es sich um das Standard-Dateisystem für OS/2

⁸⁴ [Carrier 2005, S. 282] und [Rusinovich 2009, S. 943 f.]

⁸⁵ [Carrier 2005, S. 283]

⁸⁶ [Carrier 2005, S. 359]

| Offset | Größe | Beschreibung |
|--------|-------|--------------------------------------|
| 0 | 8 | Erstellungszeitpunkt |
| 8 | 8 | Änderungszeitpunkt |
| 16 | 8 | Änderungszeitpunkt MFT-Informationen |
| 24 | 8 | Zeitpunkt des letzten Zugriffs |
| 32 | 4 | Flags ⁸⁷ |
| 36 | 4 | Maximale Anzahl Versionen |
| 40 | 4 | Versionsnummer |
| 44 | 4 | Class-ID |
| 48 | 4 | Besitzer-ID (ab Version 3.0) |
| 52 | 4 | Sicherheits-ID (ab Version 3.0) |
| 56 | 4 | Verwendete Quota |
| 64 | 8 | Update Sequence Number (USN) |

TABELLE 8: STRUKTUR DES \$STANDARD_INFORMATION-ATTRIBUTS⁸⁸

Wie Tabelle 8 zeigt, trägt das \$STANDARD_INFORMATION Attribut für die Forensik wichtige Informationen wie Erstellungs- und Änderungszeitpunkt einer Datei bzw. eines Verzeichnisses.

Das \$FILE_NAME Attribut kann zum einen für MFT-Datei-Einträge verwendet werden. In diesem Fall trägt es Informationen zum Dateinamen und dem Verzeichnis, in dem sich die Datei befindet. Eine zweite Anwendungsmöglichkeit des Attributs ist der Einsatz innerhalb eines Verzeichnis-Indizes⁸⁹, um die darin verwalteten Dateien zu referenzieren. Die Struktur eines \$FILE_NAME-Attributs wird in Tabelle 9 gegeben.

| Offset | Größe | Beschreibung |
|--------|-------|--|
| 0 | 8 | Referenz zum vorhergehenden Verzeichnis (engl. parent directory) |
| 8 | 8 | Erstellungszeitpunkt |
| 16 | 8 | Änderungszeitpunkt |
| 24 | 8 | Änderungszeitpunkt MFT-Informationen |
| 32 | 8 | Zeitpunkt des letzten Zugriffs |
| 40 | 8 | Allokierte Größe für die Datei |
| 48 | 8 | Tatsächliche Größe der Datei |
| 56 | 4 | Flags |
| 60 | 4 | Reparse-Wert |

87 Für genaue Informationen zu den Flags siehe [Carrier 2005, S. 360 f.]

88 Vgl. [Carrier 2005, S. 360] und [Russon 05, S. 7 f.]

89 Siehe Abschnitt 3.6 – Indizes für eine Beschreibung der Verzeichnis-Indizes

| | | |
|----|---|--------------------------|
| 64 | 1 | Länge des Namens |
| 65 | 1 | Namensraum ⁹⁰ |
| 66 | – | Name |

TABELLE 9: STRUKTUR DES \$FILE_NAME-ATTRIBUTS⁹¹

3.5.2.2 \$DATA

Dieses Attribut hat eine vergleichsweise einfache Struktur, da es verwendet wird, um unformatierte⁹² Dateiinhalte zu speichern. Aus diesem Grund folgen nach dem Kopf, der in „Tabelle 6: Erweiterter Kopf eines nicht-residenten Attributs“ beschrieben wird, Rohdaten, die den Inhalt der jeweiligen Datei darstellen.

Generell trägt jede herkömmliche Datei ein \$DATA-Attribut, das den Inhalt der Datei darstellt. Es ist jedoch möglich, einer Datei *weitere* \$DATA-Attribute hinzuzufügen. Geschieht dies, werden die zusätzlichen Inhalte als sog. *Alternate Data Streams* (ADS) (dt. alternative Datenströme) bezeichnet⁹³.

3.5.2.3 Weitere Attribute

Benötigt eine Datei oder ein Verzeichnis *mehrere MFT-Einträge*, beispielsweise wenn eine größere Menge Attribute angefügt werden, wird das Attribut \$ATTRIBUTE_LIST verwendet, um weitere Attribute zu referenzieren⁹⁴.

NTFS nutzt sog. Indizes, um sortierte Informationen effizient zu speichern⁹⁵. Das Attribut \$INDEX_ROOT wird verwendet, um das Wurzelement des dazu verwendeten Baums zu referenzieren. \$INDEX_ROOT ist generell resident⁹⁶. Die Nutzdaten des mit Hilfe von \$INDEX_ROOT aufgespannten Baumes werden in \$INDEX_ALLOCATION Attributen gespeichert.

90 Eine genaue Auflistung der Namensräume findet sich in [Carrier 2005, S. 363]

91 Vgl. [Carrier 2005, S. 362]

92 Aus Sicht des Dateisystems sind diese Daten unformatiert; für eine Anwendung, die diese Daten nutzt gilt dies nicht (zwangsläufig).

93 [Rusinovich 2009, S. 920 f.]. Für eine Diskussion und Beschreibung der Analyse von Alternate Data Streams siehe Abschnitt 4.2 – *Alternate Data Streams*

94 Ein MFT-Eintrag kann maximal 65536 Attribute enthalten, ist jedoch auf eine Größe von 1024 Bytes begrenzt (vgl. Einführung in 3.2). Somit kann es notwendig sein, für eine Datei weitere MFT-Einträge zu allozieren. [Carrier 2005, S. 320 f]

95 Vgl. Abschnitt 3.6 – *Indizes*

96 [Carrier 2005, S. 294]

3.6 Indizes

NTFS verwendet sog. Indizes, um *Attribute* einer Datei *sortiert* zu speichern. Somit ist es während des Lesens der Datei möglich, Inhalte schnell und zielgerichtet zu finden; dies erhöht die Arbeitsgeschwindigkeit des Dateisystems⁹⁷. Um eine Sortierung der Indizes zu erreichen werden diese in sog. *B-Bäumen* gespeichert. Hierbei handelt es sich um baumartige Datenstrukturen, die darauf basieren, dass neue Einträge in diese Datenstruktur sortiert eingefügt werden⁹⁸. Im Folgenden wird die Verwendung der Bäume exemplarisch erklärt. Eine genaue Erläuterung dieser ausgeglichenen Bäume findet sich in [Sedgewick 1992, S. 308ff].

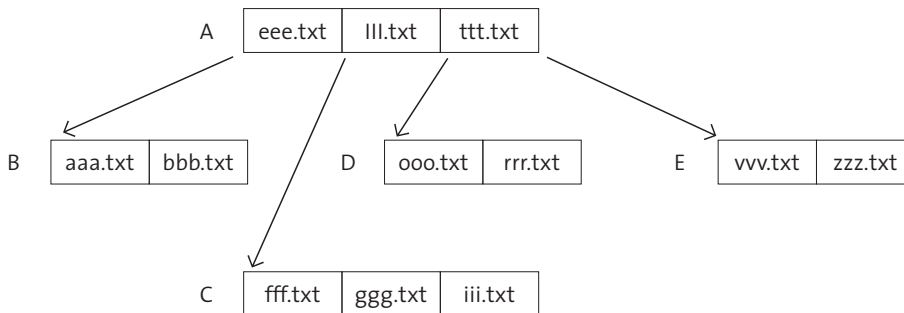


ABBILDUNG 7: EXEMPLARISCHER B-BAUM⁹⁹

Generell gelten bei der Suche in einem B-Baum folgende Regeln. Soll ein Element *E* gefunden werden, wird es mit dem *Wurzelknoten* (engl. *root node*) verglichen. Ist dessen Inhalt *größer* als der gesuchte Wert, wendet man sich dem *linken* Kindknoten (engl. *child node*) zu und verfährt dort auf die gleiche Weise. Ist jedoch der Inhalt des Wurzelknotens *kleiner* als der gesuchte Wert, wird die Suche auf den *rechten* Kindknoten verlagert. Dieses Vorgehen wird so lange wiederholt, bis entweder der Knoten mit dem entsprechenden Wert gefunden wurde oder der Baum endet. Im letztgenannten Fall befindet sich der gesuchte Wert nicht in dem Baum.

Abbildung 7 beschreibt einen B-Baum, der mit einigen Werten¹⁰⁰ gefüllt ist. An diesem Beispiel soll die Suche nach dem Wert *ggg.txt* erläutert werden¹⁰¹.

- Zu suchender Wert: *ggg.txt*
- Wurzelknoten (A): enthält *ggg.txt* nicht

97 [Carrier 2005, S. 290]

98 [Carrier 2005, S. 291]

99 Vgl. [Carrier 2005, S. 292]

100 Dabei könnte es sich z.B. um Dateinamen handeln.

101 Beispiel entnommen aus [Carrier 2005, S. 291 f]

- *ggg.txt* liegt (bei alphabetischer Sortierung) zwischen *eee.txt* und *lll.txt*
- Suche in Knoten C
- Wert *ggg.txt* befindet sich in Knoten C und wurde somit gefunden.

Das Auffinden eines Elementes ist auf diese Weise sehr einfach. Jedoch erfordert das Einfügen und Ändern einen wesentlich höheren Rechenaufwand, da möglicherweise der gesamte Baum neu strukturiert werden muss.¹⁰²

Wie eingangs erwähnt verwendet NTFS *B-Baum basierte Indizes*, um ausgewählte Attribute zu speichern. Hierzu werden zwei Strukturen als Attribute in der MFT eingetragen: `$INDEX_ROOT` und `$INDEX_ALLOCATION`. Das Attribut `$INDEX_ROOT` enthält das Wurzelement des Baumes und kann somit einen Index-Knoten tragen, der jedoch auf wenige Elemente begrenzt ist¹⁰³. `$INDEX_ALLOCATION` beinhaltet Knoten des Baumes, die in `$INDEX_ROOT` keinen Platz finden. Werden Knoten extern allokiert und mit Hilfe des `$INDEX_ROOTs` referenziert, werden sog. *Index-Records* angelegt. Typischerweise ist die Größe eines solchen Records 4096 Bytes¹⁰⁴. Da Attribute unterschiedliche Größen haben können, wird ein Record nicht zwangsläufig bis zum Ende mit Attribut-Inhalten aufgefüllt. Die genaue Belegung der Records wird mit Hilfe des Dateiattributs `$BITMAP`¹⁰⁵ festgestellt.

Die folgenden zwei Abbildungen skizzieren den Aufbau und Zusammenhang dieser Dateien.

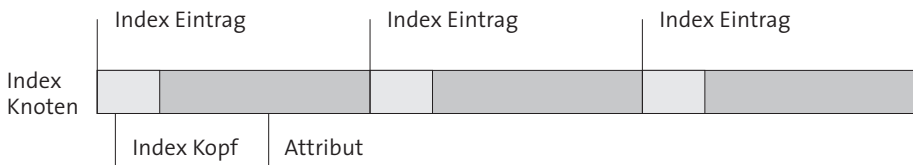


ABBILDUNG 8: STRUKTUR EINES NTFS-INDEX-KNOTENS¹⁰⁶

Die folgende Abbildung stellt den Zusammenhang zwischen MFT-Einträgen und Indizes dar und zeigt, wie Verzeichnisse mit Hilfe von Index-Records sortiert werden bzw. wie die Attribute `$INDEX_ROOT` und `$INDEX_ALLOCATION` in dem MFT-Eintrag verankert sind.

¹⁰² Vgl. [Carrier 2005, S. 291 ff]

¹⁰³ Die entspricht den Knoten und Blättern des B-Baums

¹⁰⁴ [Carrier 2005, S. 294]

¹⁰⁵ Vgl. *Tabelle 7 – Standard MFT-Attribute, ID 176*

¹⁰⁶ Vgl. [Carrier 2005, S. 294]

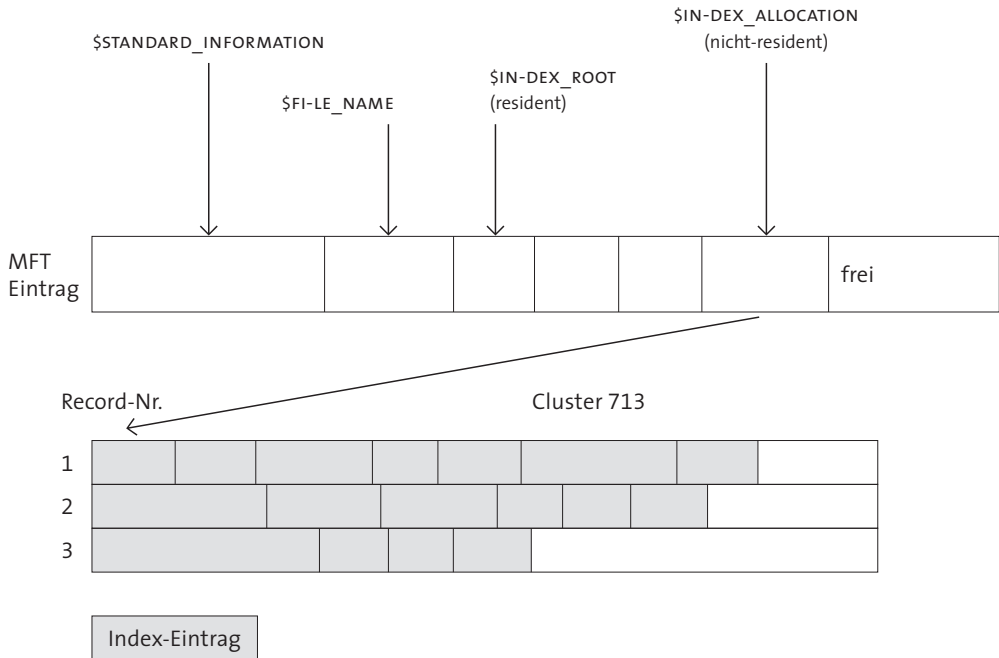


ABBILDUNG 9: EXEMPLARISCHER AUFBAU EINES MFT-EINTRAGS MIT \$INDEX_ALLOCATION¹⁰⁷

Bei NTFS-Verzeichnissen handelt es sich um *herkömmliche MFT-Einträge*, die ein spezielles Flag tragen, welches sie als Verzeichnis kennzeichnet. Ein solches Verzeichnis trägt folgende Attribute¹⁰⁸:

- \$STANDARD_INFORMATION: Informationen zu dem Verzeichnis
- \$FILE_NAME: Eigener Name und Basisinformationen
- \$INDEX_ROOT, \$INDEX_ALLOCATION: Ein Index mit \$FILE_NAME-Einträgen für die Dateien und Sub-Verzeichnisse des Verzeichnisses
- \$BITMAP: Allokationsstatus der in \$INDEX_ALLOCATION referenzierten Index-Records

Die in dem jeweiligen Verzeichnis befindlichen Dateien können mit Hilfe der Attribute \$INDEX_ROOT und \$INDEX_ALLOCATION gefunden werden.

Normale (Daten-) Dateien zeichnen sich in NTFS durch das Vorhandensein des \$DATA-Attributs aus, in dem die Inhalte der Datei gespeichert sind. Insbesondere das \$DATA-Attribut großer Dateien verwendet hierbei das Konzept der Runlists bzw. Cluster-Runs¹⁰⁹.

107 Vgl. [Carrier 2005, S. 295]

108 [Carrier 2005, S. 333]

109 Vgl. hierzu Abschnitt 3.5.1 – Externer Attribut-Inhalt: Cluster-Runs

4 Analyse

Dieses Kapitel stellt forensische Analysemöglichkeiten für das NTFS Dateisystem vor. Der erste Abschnitt beschreibt, welche Zeitinformationen zur Benutzung des Dateisystems bzw. Computers gewonnen werden können – aus Sicht der IT-Forensik eine der wichtigsten Analyseoptionen. In diesem Zusammenhang wird auf mögliche Manipulationsmöglichkeiten hingewiesen.

Abschnitt 4.2 erläutert Alternate Data Streams (ADS) und geht darauf ein, welche Möglichkeiten diese zum Verbergen forensisch relevanter Informationen haben können. Weiterhin wird beschrieben, wie Alternate Data Streams angelegt und erkannt werden können. Der nachfolgende Abschnitt beschreibt die Möglichkeit, gelöschte Dateien wiederherzustellen. Weiterhin wird auf Einschränkungen in diesem Zusammenhang eingegangen.

Das Kapitel schließt mit einem Abschnitt über verschiedene weitere Möglichkeiten und Einschränkungen der forensischen Analyse eines NTFS-Dateisystems.

4.1 Datum und Uhrzeit

Die Analyse der Datums- und Zeitstempel eines Dateisystems bzw. der darauf gespeicherten Dateien gilt als wichtiger Pfeiler forensischer Untersuchungen¹¹⁰. Auf Basis einer solchen Analyse können Tatzeiten ermittelt bzw. ein Verdacht bestätigt oder widerlegt werden. Ebenso gilt die chronologische Darstellung der Benutzeraktivitäten auf einem sog. Zeitstrahl (engl. timeline) als Mittel, die Ergebnisse einer forensischen Untersuchung zu veranschaulichen¹¹¹.

Das NTFS-Dateisystem pflegt vier verschiedene Zeitstempel, die zu diesem Zweck analysiert werden können¹¹²:

1. Änderungszeitpunkt der Datei (engl.: modification time, kurz: m-time)
2. Zeitpunkt des letzten (lesenden) Zugriffs (engl.: access time, kurz: a-time)
3. Erstellungszeitpunkt einer Datei (engl. creation time, kurz: c-time)¹¹³
4. Zeitpunkt der letzten Änderung der Metadaten einer Datei. Diese schlagen sich in einer Änderung des MFT-Eintrags nieder. Dieser Zeitstempel wird entsprechend als MFT-time beschrieben¹¹⁴

110 [Geschonnek 2004, S. 88]

111 [Geschonnek 2004, S. 151]

112 [Carrier 2005, S. 317]

113 Aufgrund der Abkürzung der ersten drei Zeitstempel werden diese zusammenfassend als MAC-Zeiten (engl. MAC times) bezeichnet.

114 Mit Hilfe der von Microsoft Windows zur Verfügung gestellten Programme (z.B. Explorer oder dir innerhalb einer Shell) ist es nicht möglich, die MFT-time zu ermitteln.

Generell werden NTFS-Zeiten als verstrichene Einhundert-Nanosekunden-Anteile seit dem 01.01.1601 UTC gespeichert¹¹⁵. Anpassungen an lokale Zeitzonen oder die Berücksichtigung der Unterschiede zwischen Sommer- und Winterzeiten werden nach dem Lesen der UTC-Zeit durch das Betriebssystem vorgenommen¹¹⁶.

Das NTFS-Dateisystem sieht zwei verschiedene Stellen zum Speichern der oben genannten Zeitinformationen vor. Diese befinden sich in den Attributen \$STANDARD_INFORMATION¹¹⁷ und \$FILE_NAME¹¹⁸, die jedoch während des Lebenszyklus einer Datei nicht gleichermaßen gepflegt werden. Die Informationen in \$FILE_NAME werden nach Carrier und Sammes lediglich beim Anlegen oder Verschieben einer Datei geschrieben¹¹⁹. Die Anzeige der Dateiinformationen innerhalb des Windows Explorers und die Anzeige von Zeitstempel forensischer Software stützt sich auf die Angaben des Attributs \$STANDARD_INFORMATION, die bei jeder für die Zeitstempel relevanten Aktion gepflegt werden¹²⁰.

Während der Analyse eines NTFS-Dateisystems sollten jedoch auch Manipulationsmöglichkeiten bzw. diverse Eigenarten des Dateisystems in Betracht gezogen werden. Diese werden nachfolgend erläutert.

Aus Geschwindigkeitsgründen (z.B. für Datei-Server mit hoher Last) ist es möglich, das Schreiben der a-time, also des Zeitpunkts des letzten (lesenden) Zugriffs eines NTFS-Dateisystems abzuschalten¹²¹. Dies hat Einfluss auf die Analysemöglichkeiten, da der Zeitpunkt des letzten Zugriffs einzelner Dateien nicht bestimmt werden kann. Weiterhin verwendet Windows einen Cache für die Zeitstempel des letzten Zugriffs (a-time). Im Fall eines abrupten Ausschaltens des Computers (z.B. Stromausfall) kann es vorkommen, dass nicht alle Zeiten auf die Festplatte geschrieben werden können. Auch dies schränkt, jedoch in geringerem Maße, die Analyse eines solchen Datenträgers ein¹²².

Einem Angreifer oder einem Täter, der vorsätzlich Spuren verwischen möchte, ist es mit frei verfügbaren Programmen möglich, die MAC-Einträge des Dateisystems zu manipulieren¹²³. Dieser Fakt sollte bei einer forensischen Untersuchung ebenso wie die im vorigen Absatz genannten möglichen Einschränkungen bedacht werden.

115 [Carrier 2005, S. 317]. Vgl. auch Abschnitt 3.1 – *Überblick*

116 [Carvey 2007, S. 230]

117 Siehe *Tabelle 8: Struktur des \$STANDARD_INFORMATION-Attributs, Offset 0–24*

118 Siehe *Tabelle 9: Struktur des \$FILE_NAME-Attributs, Offset 8–32*

119 [Carrier 2005, S. 360] und [Carrier 2005, S. 325] sowie [Sammes 2007, S. 271]

120 [Sammes 2007, S. 271]

121 [Carvey 2007, S. 230].

122 [Carrier 2005, S. 326]

123 [Carvey 2007, S. 231–232]

4.2 Alternate Data Streams (ADS)

Alternate Data Streams beschreiben die Möglichkeit des NTFS-Dateisystems, mehrere \$DATA-Attribute einer Datei zuzuordnen und damit mehrere Dateien unter einem einzelnen Dateinamen zu speichern¹²⁴. Die Besonderheit dieser alternativen Datenströme besteht darin, dass sie von herkömmlichen Mitteln des Betriebssystems nicht angezeigt, aber angelegt und verwendet werden können. Lediglich seit Windows Vista existiert eine Möglichkeit, ohne die Verwendung zusätzlicher Programme, alternative Datenströme anzuzeigen¹²⁵.

Das Ansprechen eines ADS erfolgt durch die Nennung der Trägerdatei¹²⁶ sowie, mit Hilfe eines Doppelpunktes abgetrennt, die Nennung des ADS (Name eines weiteren \$DATA Attributs):

`Dateiname.ext:ADSName`

Das folgende Beispiel demonstriert die Verwendung verschiedener Datenströme unter Windows sowie die Anzeige der Alternate Data Streams mit Hilfe des *dir* Kommandos (Schritt 5):

1. Erstellen der Datei Datei01.txt:
`C:\ADS-test>echo Datei01-Inhalt >Datei01.txt`
2. Manuelles Erstellen eines Alternate Data Streams:
`C:\ADS-test>echo DasIstEinADS >Datei01.txt:MeinADS01`
3. Kopieren einer ausführbaren Datei in einen Alternate Data Stream:
`C:\ADS-test>type c:\windows\system32\calc.exe >Datei01.txt:CalcADS.exe`
4. Herkömmliche Anzeige der Verzeichnisinhalte¹²⁷
`C:\ADS-test>dir`

```
Volume in drive C has no label.  
Volume Serial Number is XXXX-XXXX
```

```
Directory of C:\ADS-test
```

```
25.08.2009  22:21    <DIR>          .  
25.08.2009  22:21    <DIR>          ..
```

124 Vgl. Abschnitt 3.5.2.2 – \$DATA

125 [Carvey 2007, S. 242]

126 Bezeichnet gleichzeitig das erste \$DATA-Attribut einer Datei.

127 Der Windows Explorer zeigt Alternate Data Streams ebenfalls nicht an

```

25.08.2009  22:23                17 Datei01.txt
              1 File(s)                17 bytes
              2 Dir(s)  31.228.973.056 bytes free

```

5. Anzeigen des Verzeichnisses mit Alternate Data Streams (ab Windows Vista)

```
C:\ADS-test>dir /R
```

```

Volume in drive C has no label.
Volume Serial Number is XXXX-XXXX

```

```
Directory of C:\ADS-test
```

```

25.08.2009  22:21    <DIR>      .
25.08.2009  22:21    <DIR>      ..
25.08.2009  22:23                17 Datei01.txt
              776.192 Datei01.txt:CalcADS.exe:$DATA
              15 Datei01.txt:MeinADS01:$DATA
              1 File(s)                17 bytes
              2 Dir(s)  31.228.973.056 bytes free

```

6. Ausführen einer Datei als ADS

```
C:\ADS-test\Datei01:CalcADS.exe
```

Im Rahmen einer forensischen Analyse eines mit NTFS formatierten Datenträgers sollten alternativen Datenströmen besondere Aufmerksamkeit zukommen, da diese für potenzielle Täter als Verstecke dienen können¹²⁸. Jedoch verwenden nach Erfahrung des Autors das Windows Betriebssystem und andere Programme, wie beispielsweise Antivirus-Lösungen zunehmend Alternate Data Streams, um Metainformationen¹²⁹ mit Dateien zu verbinden, so dass das Vorhandensein eines ADS nicht zwangsläufig auf nicht bestimmungsgemäße Handlungen hindeutet¹³⁰.

4.3 Wiederherstellen gelöschter Dateien

Gelöschte Dateien stellen aus forensischer Sicht eine interessante Quelle dar, da potenzielle Täter häufig nicht wissen, dass diese unter bestimmten Umständen wieder hergestellt werden können¹³¹. Dieser Abschnitt beschreibt das Wiederherstellen direkt gelöschter Dateien. Eine Wiederherstellung auf Basis des sog. Papierkorbs, der einen Zwischenspeicher für gelöschte Dateien darstellt, kann Carvey¹³² entnommen werden.

128 [Steel 2006, S. 76], [Carvey 2007, S. 242], [Geschonneck 2004, S. 91]

129 Windows verwendet ADS beispielsweise, um Metainformationen über MP3- oder JPEG-Dateien zu speichern. Antivirus-Lösungen verankern mit Hilfe von alternativen Datenströmen z.B. Checksummen der Dateien.

130 Siehe hierzu auch [Sammes 2007, S. 415-419]

131 [Geschonneck 2004, S. 96]

132 [Carvey 2007, S. 221-224]

Das Wiederherstellen gelöschter Dateien auf einem NTFS-Datenträger gilt als relativ einfach¹³³, da während des Löschvorgangs lediglich ein Byte innerhalb des MFT-Eintrags verändert wird; sämtliche anderen Informationen bleiben erhalten¹³⁴. Dennoch sollten bei der Analyse gelöschter Dateien einige Einschränkungen beachtet werden; diese werden im Folgenden beschrieben.

Das Löschen einer Datei oder eines Verzeichnisses führt zu einer Änderung des Flags der jeweiligen Datei innerhalb des MFT-Eintrags¹³⁵. Dieser zwei Bytes große Wert kann folgende Informationen tragen und gibt Aufschluss über die Verwendung eines MFT-Eintrags¹³⁶:

- 00 00: gelöschter Dateieintrag
- 01 00: verwendeter Dateieintrag
- 02 00: gelöschter Verzeichniseintrag
- 03 00: verwendeter Verzeichniseintrag

Mit Hilfe der weiterhin in dem File-Record (MFT-Eintrag) vorhandenen Informationen kann die Datei wiederhergestellt werden, da auch die Referenzen auf mögliche externe Attribute und damit verbundene Cluster-Runs nicht gelöscht werden. Hierbei gelten jedoch Einschränkungen:

Zwar trägt das Attribut `$FILE_NAME` eine Referenz¹³⁷ auf das Verzeichnis, in dem die Datei lag. Es kann jedoch nicht sichergestellt werden, dass es sich dabei um das gleiche Verzeichnis handelt, wie zum Erstellungszeitpunkt der Datei, sollte der gleiche MFT-Record für eine neues Verzeichnis verwendet worden sein¹³⁸. Weiterhin wird während des Löschvorgangs die Referenz auf die gelöschte Datei innerhalb des Verzeichnisses entfernt, in dem sie sich befand. Dies kann dazu führen, dass der gesamte Inhalt dieses Records auf Basis eines B-Baum neu sortiert wird; auf diese Weise gehen Informationen über das Vorhandensein der gelöschten Datei (für dieses Verzeichnis) verloren¹³⁹. Diese Informationen betreffen jedoch nicht die Datei selbst, sondern ihr Lage innerhalb des Dateisystems (Pfadangabe).

Weiterhin ist es möglich, dass alle oder einzelne Cluster der gelöschten Datei mit neuen Inhalten überschrieben werden. In diesem Fall ist es lediglich möglich, die verbliebenen, nicht überschriebenen Teile wiederherzustellen¹⁴⁰.

133 [Carrier 2005, S. 348]

134 [Sammes 2007, S. 272], [Carrier 2005, S. 348]. Weiterhin wird die Datei `$Bitmap` angepasst, um die Cluster der Datei als „Frei“ zu markieren [Steel 2007, S. 228].

135 Siehe hierzu *Tabelle 1: Struktur eines MFT-Eintrags (file record), Offset 22*

136 [Steel 2007, S. 236]

137 Vgl. *Tabelle 9: Struktur des \$FILE_NAME-Attributs, Offset 0*

138 [Sammes 2007, S. 273]

139 Vgl. hierzu Abschnitt über B-Bäume in Kapitel 3,6 – *Indizes* sowie [Carrier 2005, S. 348].

140 [Carrier 2005, S. 248]

Es kann jedoch nicht davon ausgegangen werden, dass sämtliche auf NTFS-Datenträgern gespeicherten Dateien wiederhergestellt werden können, wenn die MFT-Einträge nicht wiederverwendet wurden. Spezielle Programme, freie sowie kommerzielle, sind in der Lage, Dateien rückstandslos zu löschen. Mit Hilfe solcher Programme können Angreifer bzw. andere Täter Dateien löschen, ohne dass diese wiederhergestellt werden können.¹⁴¹

4.4 Weitere Analysemöglichkeiten

Im Rahmen der Analyse eines NTFS formatierten Datenträgers ist es notwendig, die Eigenheiten des Dateisystems sowie verschiedene Analysemethoden zu kennen. Dieser Abschnitt zeigt weitere aus forensischer Sicht wichtige Aspekte auf und diskutiert diese mit Bezug auf eine Analyse.

Die Analyse des sog. SchlupfSpeichers (engl. slack space) gilt als wichtiges Element einer forensischen Untersuchung¹⁴². Sogenannter Drive-Slack entsteht, wenn die Größe einer Datei nicht dem Vielfachen der Größe einer Zuordnungseinheit des Dateisystems (in Fall von NTFS: Cluster) entspricht¹⁴³. In diesem Fall ist der letzte für die Datei allokierte Cluster nicht vollständig mit Inhalten der neu angelegten Datei gefüllt. Inhalte früherer dort gespeicherter Dateien bleiben bestehen und können analysiert werden. Da Festplatten nicht byteweise, sondern lediglich zu Blöcken a 512 Bytes beschrieben werden können, kann es vorkommen, dass zwischen dem Ende einer Datei und dem Ende des zuletzt beschriebenen Sektors ein Bereich entsteht, der nicht von der Datei verwendet wird, aber aufgrund der zuvor genannten Restriktion geschrieben werden muss. Dieser Bereich wird, da frühere Microsoft-Betriebssysteme dazu zufällig gewählte Speicherinhalte verwendeten, RAM-Slack genannt. Aktuelle Windows-Versionen füllen diesen Bereich mit Nullen¹⁴⁴. Somit ist eine gewinnbringende Analyse des RAM-Slacks aktueller Windows-Versionen nicht gegeben. Folgende Abbildung stellt den Zusammenhang zwischen Cluster- bzw. Sektorbelegung und Slack-Space dar:

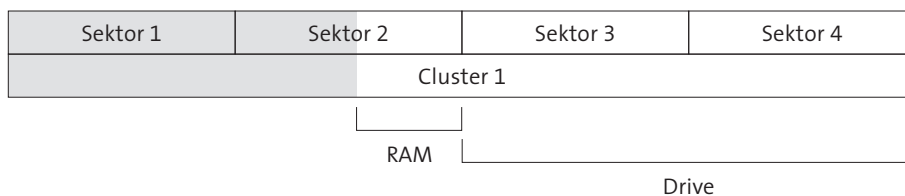


ABBILDUNG 10: RAM- UND DRIVE-SLACK EINES NTFS-DATEISYSTEMS (CLUSTERGRÖSSE: 2048 BYTES)

141 [Geschonneck 2004, S. 97]

142 [Steel 2006, S. 52], [Carrier 2005, S. 187–188], [Geschonneck 2004, S. 85]

143 [Carrier 2005, S. 187]

144 [Geschonneck 2004, S. 86], [Carrier 2005, S. 187]

In NTFS kann Schlupfspeicher weiterhin entstehen, wenn Einträge der \$MFT-Datei wieder- bzw. weiterverwendet werden¹⁴⁵. Dieser Typ Schlupfspeicher wird als MFT-Slack bezeichnet. Er entsteht, wenn Dateien aufgrund ihrer geringen Größe zunächst resident¹⁴⁶ gehalten werden und somit der Inhalt des \$DATA-Attributs¹⁴⁷ in der MFT selbst zu liegen kommt. Wird danach der Datei weiterer Inhalt hinzugefügt, überschreitet diese die maximale Dateigröße, die resident gehalten werden kann und wird daher als nicht-residentes Attribut mit Hilfe von Cluster-Runs geführt. Der Verweis auf diese Cluster-Runs kann jedoch weniger Platz innerhalb des MFT-Eintrags beanspruchen, als vorher durch den Inhalt der Datei, also das residente \$DATA-Attribut belegt wurde. Ist dies der Fall, bleiben die vorher dort gespeicherten Informationen in dem MFT-Eintrag erhalten und können (nach der Ende-Markierung des MFT-Eintrags) ausgelesen werden.

Ein weiterer Aspekt, der bei der Analyse eines NTFS-Dateisystems bedacht werden sollte, ist die Möglichkeit, in NTFS Dateien und Verzeichnissen Rechte zuzuweisen¹⁴⁸. Dies ist insbesondere in Umgebungen relevant, in denen mehrere Benutzer auf dem gleichen Datenträger arbeiten. Hier sollte geprüft werden, in wie weit ein Beschuldiger auf Basis dieser Rechte in der Lage war, auf fragliche Dateien zuzugreifen bzw. diese zu erstellen. Von der aktuellen, im Rahmen des forensischen Abbildes erhaltenen Einstellung der Rechte kann jedoch nicht abgeleitet werden, dass diese sich zu jeder Zeit in dem vorgefundenen Zustand befanden; aus diesem Grund kann die vorgefundene Rechtestruktur nach Erfahrung des Autors lediglich als Anhaltspunkt bzw. Indiz, nicht jedoch als Beweis gelten.

Da die Prüfung der in NTFS verankerten Rechte gegen die des aktiven Benutzers während des Lesens oder Schreibens dem jeweiligen Betriebssystem obliegt, ist es möglich, mit Hilfe eines anderen Computers (z.B. durch Umbauen der Festplatte) oder durch Verwenden eines alternativen Betriebssystems beliebige Dateien auf dem Datenträger zu manipulieren oder zu lesen¹⁴⁹. Dies kann nur durch die Verwendung einer entsprechenden Verschlüsselung verhindert werden, die jedoch im Rahmen dieser Arbeit nicht erläutert wird.

5 Fazit und Ausblick

Das mit Windows NT eingeführte Dateisystem NTFS bietet eine Reihe von Funktionen, die sein Vorgänger, das File Allocation Table (FAT) Dateisystem nicht zur Verfügung stellte, jedoch von Anwendern und Betriebssystem zunehmend erwartet wur-

145 [Sammes 2007, S. 273–275]

146 Siehe hierzu Abschnitt 3.5 – *Attribute*

147 Das \$DATA-Attribut beschreibt den Inhalt einer Datei; vgl. hierzu Abschnitt 3.5.2.2 – *\$DATA*

148 [Sammes 2007, S. 217]

149 [Steel 2006, S. 84 f.]

den. Dazu werden beispielsweise eine größere Robustheit gegenüber Abstürzen und Hardwaredefekten sowie die Möglichkeit einer Rechteverwaltung bzw. Rechteeinschränkung einzelner Benutzer oder Gruppen gerechnet. Darüber hinaus bietet NTFS die Möglichkeit, Dateien zu komprimieren und zu verschlüsseln. Die letztgenannten Funktionalitäten wurden jedoch innerhalb dieser Arbeit nicht vollumfänglich dargestellt; vielmehr wurde ein Überblick über das Dateisystem bzw. seine wichtigsten Komponenten und deren Zusammenhänge sowie forensische Analysemöglichkeiten gegeben.

Kapitel 2 stellte zu diesem Zweck zunächst die Verschiedenen Versionen des Dateisystems sowie seine erweiterten Funktionalitäten wie Kompression und Verschlüsselung vor. Kapitel 3 gab einen Überblick über die Grundsätze des Dateisystems, verschiedene Standarddateien sowie Attribute, die verwendet werden, um Dateien und Verzeichnisse darzustellen. Kapitel 4 schließlich stelle forensische Analysemöglichkeiten vor. Hierbei wurden die Zeitstempel des Dateisystems und die Möglichkeiten der Wiederherstellung gelöschter Dateien beleuchtet. Weiterhin wurden sog. Alternate Data Streams (ADS) beschrieben, die mit Hilfe des Betriebssystems relativ schwer zu finden sind und deshalb von Tätern als Daten-Versteck genutzt werden.

Aus forensischer Sicht ist die Dokumentation des NTSF-Dateisystems seitens Microsoft unzureichend um eine umfassende und genaue Analyse durchführen zu können. Insbesondere das Fehlen genauer Spezifikation führt dazu, dass sich ein Großteil der Aussagen über Funktionsweise und Aufbau des Dateisystems auf umfangreichen Untersuchungen einzelner Autoren sowie dem Reverse Engineering der Microsoft-Treiber beruht. Einzelne Unternehmen sind jedoch in der Lage, bei Unterzeichnung einer Geheimhaltungserklärung, die Spezifikationen einzusehen¹⁵⁰. Insbesondere eine vollumfängliche Dokumentation des Dateisystems im Rahmen wissenschaftlicher Arbeit ist auf dieser Basis nicht möglich.

Weiterhin ist davon auszugehen, dass eine vollständige und freie Spezifizierung des Dateisystems auch aus Sicht der IT-Forensik Vorteile bieten würde. Beispielsweise sind forensische Ansätze bzgl. der Journaling-Funktionalität vorhanden, jedoch führen verschiedene Unklarheiten, beispielweise bzgl. der Vorhaltezeit der Journale, für Unsicherheiten innerhalb einer Analyse¹⁵¹.

Insgesamt kann jedoch auf Basis in dieser Arbeit vorgestellter Datenstrukturen und -typen sowie der Erläuterung der Analyseansätze eine valide forensische Analyse durchgeführt werden.

150 Dies führt jedoch aus Sicht des Autors dazu, dass auf einer solchen Basis entstandene Produkte zu relativ hohen Preisen verfügbar sind.

151 [Carrier 2005, S. 343]

6 Quellen

- [Carrier 2005] Carrier, B.: File System Forensic Analysis, Addison-Wesley, Crawfordsville, 2005
- [Carvey 2007] Carvey, H.: Windows Forensic Analysis, Syngress Publishing, Burlington, 2007
- [Geschonnek 2004] Geschonnek, A.: Computer Forensik, dpunkt.verlag, Heidelberg 2004
- [Steel 2006] Steel, C.: Windows Forensics. The Field Guide for Conducting Corporate Computer Investigations, Wiley Publishing, Indianapolis 2006
- [Sammes 2007] Sammes, T., Jenkinson, B.: Forensic Computing, 2nd Edition, Springer, London 2007
- [Rusinovich 2009] Rusinovich, M. E., Solomon, D.: Windows Internals. , 5th edition, Microsoft Press, Redmond, Washington 2009
- [Russon 05] Russon, R., Fledel, Y.: NTFS Documentation, Version 0.5, The Linux NTFS Project, <http://data.linux-ntfs.org/ntfsdoc.pdf>, Zugriff am 07.07.2009
- [Sedgewick 1992] Sedgewick, R.: Algorithmen in C, Addison-Wesley, Bonn; München; Paris [u.a.], 1992

Ebenfalls erhältlich:

Band 1: Metastudie Open-Source-Software und ihre Bedeutung für Innovatives Handeln

Band 2: Studie zum Innovationsverhalten deutscher Software-Entwicklungsunternehmen

Band 3: Erfolgskriterien für Identifizierungs-, Authentifizierungs- und Signaturverfahren auf Basis asymmetrischer kryptographischer Verfahren (EKIAS)

© 2011 Eigenverlag, Berlin

Satz: Martin Schüngel

Druck: digital business and printing GmbH, D-10409 Berlin

ISSN 1863-5016