Forensische Analyse komplexer Unix-Dateisysteme

Knut Eckstein
NC3A, P.O. Box 174, Den Haag, NL
knut@acm.org

Abstract: Moderne Dateisysteme stellen effiziente Verfahren zur Verwaltung extrem großer Datenmengen bereit, die in komplexen, vernetzten Umgebungen anfallen. Der vorliegende Beitrag untersucht die Auswirkungen von Eigenschaften moderner Unix-Dateisysteme auf Techniken der forensischen Analyse und führt zu zwei Schlußfolgerungen: Höhere Volatilität dynamischer Verwaltungs-Datenstrukturen könne die forensische Analyse einerseits erschweren. Andererseits erlauben komplexere und neuartige Verwaltungs-Datenstrukturen zusätzliche forensische Informationsgewinne.

1 Einleitung

Den Anforderungen immer komplexer werdender Systeme genügen "klassische" Unix-Dateisysteme, die auf dem Berkeley Fast File System (FFS) beruhen, nicht mehr. Moderne Unix-Implementierungen setzen seit einiger Zeit vermehrt Dateisysteme ein, die sich grundlegend vom FFS-Design unterscheiden, beispielsweise durch Journale, dynamische Datenstrukturen und die Allozierung von Dateien in große Datenblockgruppen.

Die forensische Analyse FFS-basierter Dateisysteme ist in einer Reihe von Arbeiten eingehend erforscht worden [FV04, Car05]. Zu den Aufgaben der forensischen Analyse von Dateisystemen zählt das Wiederherstellen gelöschter Dateien, das Auffinden vorsätzlich versteckter Informationen sowie als Grundsatzaufgabe das Nachvollziehen möglichst aller Angriffsschritte während eines Systemeinbruchs.

Dieser Artikel untersucht die Auswirkungen einer Reihe von Eigenschaften komplexer Dateisysteme auf deren forensische Analyse. Die Gliederung orientiert sich an etablierten Abstraktionsebenen der digitalen Medienforensik: Abschnitt 2 diskutiert Aspekte innerhalb der Medienmanagement-Ebene, während Abschnitt 3 die verschiedenen Kategorien der Dateisystem-Ebene behandelt. Abschnitt 4 enthält Zusammenfassung und Ausblick.

2 Medienmanagement-Ebene

Der Begriff Medienmanagement steht für die Verwaltung und Aufteilung eines Datenträgers in Abschnitte oder Partitionen. Im herkömmlichen Fall ist jedem Abschnitt genau ein Dateisystem zugeordnet.

Diese 1:1-Zuordnung ist in einigen neueren Dateisystemen wie *JFS* und *Veritas FS* aufgehoben, die in einem Datenträgerabschnitt mehrere Dateisysteme enthalten können [Pat03]. Dies beeinträchtigt eine Standardmethode der forensischen Analyse, bei der alle als unbelegt markierten Datenblöcke ausgelesen werden, um gelöschte oder versteckte Dateien aufzuspüren. Bei den oben genannten Dateisystemen können die nicht benutzten Datenblöcke unter Umständen keinem der in der Partition enthaltenen Dateisysteme eindeutig zugeordnet werden, was die Rekonstruktion gelöschter Dateien erschwert.

3 Dateisystem-Ebene

Ein Dateisystem selbst läßt sich nach [Car05] zur forensischen Analyse in vier Kategorien unterteilen: Dateisystem, Datenblock, Metadaten und Namen.

3.1 Dateisystem-Kategorie

Diese Kategorie beinhaltet für klassische Unix-Dateisysteme in der Regel den Superblock, in dem wesentliche globale Parameter abgelegt werden. Diese Datenstruktur kann bei modernen Dateisytemen erheblich komplexer ausfallen[1]. Aus diesem Umstand resultieren steigende Anforderungen an forensiche Werkzeuge, diese Datenstrukturen auf Korrektheit und eventuell versteckte Daten zu untersuchen.

Eine wesentliche Erweiterung dieser Kategorie stellt das *Journal* eines Dateisystems dar. Es befindet sich entweder in einer Systemdatei im Dateisystem selbst oder in einer separaten Partition. Das Journal stellt eine erhebliche forensische Informationsquelle dar, deren Nutzung jedoch die Erweiterung bestehender forensischer Analyseprogramme voraussetzt. Durch die Auswertung des Journals erhält der Forensiker Einblick in Veränderungen von Daten oder Metadaten in dem vom Journal abgedeckten Zeitraum, unabhängig vom aktuellen Inhalt des untersuchten Dateisystems, beispielsweise zur Rekonstruktion von Daten, die ein Angreifer zum Verwischen seiner Spuren durch ein Überschreibwerkzeug vermeintlich sicher gelöscht zu haben glaubt.

Die forensische Aussagekraft hängt neben der Journalgröße auch vom Journaltyp ab: Während blockorientiere Journalimplementierungen stets ganze Datenblöcke enthalten, die einer Veränderung unterliegen, speichern transaktionsorientierte Journale allein den Änderungsumfang. Aus forensischer Sicht bieten erstere den Vorteil, in von der eigentlichen Transaktion unberührten Blockabschnitten Inhalte abzuspeichern, die eventuell zu einem späteren Zeitpunkt gelöscht werden. Transaktionsorientierte Journale dagegen benötigen definitionsgemäß weniger Platz und können so bei bei gleichbleibender Journalgröße weiter in die Vergangenheit zurückreichen.

[1] Beispielsweise durch Mechanismen, die die in Abschnitt 2 beschriebene Verwaltung mehrerer Dateisysteme in einer Partition ermöglichen.

3.2 Datenblock-Kategorie

Diese Kategorie beinhaltet die Datenstrukturen zur Verwaltung aller adressierbaren logischen Datenblöcke eines Dateisystems.

Eine der wichtigsten Verwaltungsstrukturen in dieser Kategorie, die nahezu alle Dateisysteme verwenden, ist die so genannte "Block Allocation Bitmap". Diese Liste bildet für klassische Dateisysteme die Grundlage für eine von mehreren extensiven Konsistenzprüfungen durch das Programm *fsck*: Ein Bit in diesem Feld darf nur dann gesetzt sein, wenn der zugehörige Datenblock von genau einer Datei referenziert wird. Eine diesbezügliche Prüfung des Dateisystems (z. B. nach einem Systemabsturz) kann den Bootvorgang empfindlich verlängern, bei hochkapazitiven Serversystemen bis zu mehreren Stunden.

Um dies zu umgehen, wird bei Journal-basierten Dateisystemen die Konsistenzprüfung auf die Überprüfung des Journals reduziert. Da dieses per Definition alle vor dem Systemabsturz getätigten Änderungen enthält, können so eventuelle Inkonsistenzen sehr schnell lokalisiert und behoben werden. Die Vollständigkeit dieser Konsistenzprüfung gilt jedoch nur unter der Annahme, dass alle Änderungen an der Block Allocation Bitmap im Journal aufgezeichnet werden. Genau diese Annahme kann ein Angreifer, der über Systemprivilegien verfügt, unterlaufen und dadurch weitestgehend unbemerkt große Datenmengen in einem solchen, komplexen Dateisystem verstecken [EJ05].

3.3 Metadaten-Kategorie

Metadaten beschreiben Eigenschaften von Dateien. Diese Strukturen werden in der Unix-Welt "Inodes" genannt und enthalten Typ, Größe, Zeitstempel, Zugriffsrechte und Datenblockreferenzen jeweils einer Datei (nicht jedoch deren Namen, siehe Abschnitt 3.4). Insbesondere die Zeitstempel sind von forensischem Interesse, da deren Auswertung eine "Zeitleiste" eines Systems generiert, aus der die Abfolge der Handlungen eines Angreifers abgelesen werden kann.

Klassische Unix-Dateisysteme referenzieren, wie in Abbildung 1 links dargestellt, die Datenblöcke, die sie einer Datei zuordnen, durch individuelle Verweise in Form der Adresse des Datenblocks im Dateisystem. Die Position eines Datenblocks in einer Datei ist implizit durch die Position der Referenz in der Gesamtliste der Referenzen gegeben. Mit steigender Dateigröße werden zusätzlich einfach, zweifach und dreifach indirekte Verweise eingesetzt, da Dateigrößen von mehreren Gigabyte ca. 10^6 Verweise auf Datenblöcke der Standardgröße 4 Kilobyte benötigen.

Eine Reihe von aktuellen Dateisystemen führt anstelle der Adressierung einzelner Datenblöcke die Adressierung von Datenblockabschnitten ("extents") ein. Dazu werden beim JFS Dateisystem, wie in Abbildung 1 rechts zu sehen, statt Datenblöcken fester Länge Tripel (O,A,L) referenziert. Darin sind in denen die physikalische Startadresse des Datenblockabschnitts im Speichermedium (A), die logische Offsetadresse des Abschnitts in der abzubildenden Datei (O) und die Länge des Abschnitts (L) abgelegt.

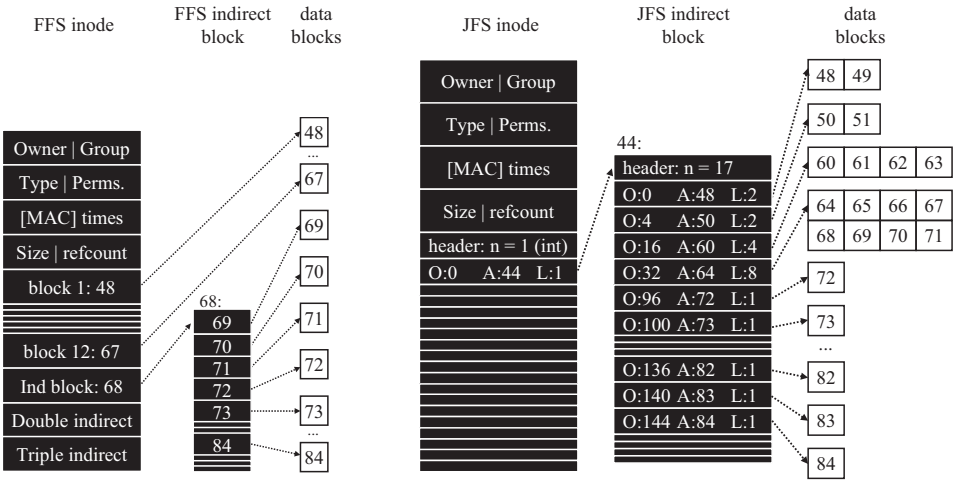

Abbildung 1: Vergleich der zur Dateiallokierung eingesetzten Datenstrukturen

Aus Sicht der forensischen Analyse führt der Einsatz von Datenblockabschnitten zu Dateien, die auf dem Speichermedium örtlich eher zusammenhängend untergebracht werden können. Dies erleichtert die Wiederherstellung einer gelöschten Datei in Maße. Lassen sich nur Fragmente einer gelöschten Datei wiederfinden, so können die Tripel (O,A,L) in einem identifizierten "JFS indirect block" durch ihre Offsetadressen dazu verwendet werden, Datenblöcke den Abschnitten der gelöschten Datei eindeutig zuzuordnen. Im Vergleich mit dem klassischen Adressierungsschema, wo der "FFS indirect block" nur aus weiteren Blockadressen besteht, zeigt sich der höhere forensische Gehalt der komplexeren Verwaltungs-Datenstruktur deutlich.

Neben der Verwaltung von Datenblockreferenzen unterscheiden sich klassische und moderne Dateisysteme auch bezüglich der Speicherung der Metadaten selbst. In FFS und verwandten Systemen werden für Inodes zum Zeitpunkt der Formatierung Bereiche auf dem Datenträger unveränderlich reserviert. Daher werden typischwerweise sehr viel mehr Inodes angelegt, als je gleichzeitig gebraucht werden. Diese Umstände führen dazu, dass bei der forensischen Analyse von FFS-basierten Dateisystemen häufig noch Metadaten gefunden werden, deren Löschung bereits mehrere Jahre zurückliegt.

In aktuellen Dateisystemen werden Inodes zur Erhöhung der Flexibilität dynamisch generiert und platziert. Dies bedeutet einerseits, dass Inodes stets nur soviel Platz einnehmen, wie benötigt wird. Andererseits können gelöschte Inodes grundsätzlich durch beliebige andere Nutzdaten überschrieben werden, woraus sich unmittelbar die These der erhöhten Volatilität forensisch wertvoller Informationen in komplexen Dateisystemen ableitet.

Zur Vorbereitung dieses Artikels wurde versucht, die erhöhte Volatilität von JFS Inodes durch Vergleichsmessungen nachzuweisen. Während klar erkennbar war, dass die Gesamtzahl der JFS Inodes um mindestens eine Größenordnung unter der vergleichbarer FFS Dateisysteme liegt, wurde bei der weiteren Auswertung deutlich, dass belastbare Verglei-

che die Einhaltung einer Reihe von Bedingungen erfordern, die für die vorliegenden Meßdaten nicht alle gegeben waren: Neben gleichem Alter und gleichem Nutzungsverhalten müssen außerdem gleiche physikalische Größe und ein "Mindestalter" der Dateisysteme gegeben sein, um Volatilitätsunterschiede zweifelsfrei belegen zu können.

3.4 Namens-Kategorie

Die Kartegorie der Namen in Dateisystemen umfasst Datenstrukturen zur Zuordnung von Dateinamen und Inodes sowie zur Verwalten von Namen in Verzeichnissen. Aktuelle Dateisysteme setzen komplexere Verwaltungsstrukturen ein, um zum Beispiel Dateinamen in sortierten Bäumen zur Verringerung der Suchzeiten zu verwalten. Diese Baumstrukturen können wie in Unterabschnitt 3.3 zusätzliche, forensisch nutzbare Hinweise enthalten, beispielsweise Sortierschlüssel in Baumknoten im Falle von JFS.

Bei JFS kann das Journal eingesetzt werden, um mit der Relation Inode–Name die entscheidende forensische Information dieser Kategorie wieder zu ermitteln, welche JFS beim Löschen eines Verzeichniseintrags überschreibt [Eck04].

4 Zusammenfassung und Ausblick

Im vorliegenden Beitrag wurden relevante Eigenschaften moderner Dateisysteme vorgestellt und deren Auswirkungen auf die forensische Analyse untersucht. Komplexere Dateisysteme erzeugen Datenstrukturen von höherem forensischen Wert, wie gezeigt im Journal und bei der "extent"-basierten Allozierung von Dateien.

Zugleich kann die erhöhte Variabilität und Dynamik forensisch wertvoller Verwaltungs-Datenstrukturen, möglicherweise in Abhängigkeit von Betriebssystemimplementierungen und Lastzuständen, zu erhöhter Volatilität dieser Strukturen führen. Der experimentelle Nachweis einer erhöhten Volatilität erfordert ein sorgfältig durchdachtes Messverfahren unter Berücksichtigung verschiedener Lastsituationen.

Literatur

[Car05] B. Carrier. *File System Forensic Analysis*. Pearson Education, 2005.

[Eck04] K. Eckstein. Forensics for Advanced Unix File Systems. In *Proceedings of the 2004 IEEE Workshop on Information Assurance*, USMA West Point, NY, Juni 2004.

[EJ05] K. Eckstein und M. Jahnke. Data Hiding in Journaling File Systems. Eingereicht beim Digital Forensic Research Workshop, 2005.

[FV04] D. Farmer und W. Venema. *Forensic Discovery*. Pearson Education, Dezember 2004.

[Pat03] S. Pate. *UNIX File Systems, Evolution, Design and Implementation*. Wiley, 2003.