Operation System Security

05 – Dateisysteme

SECURNITE

OSSEC 05 - Dateisysteme

Einführung in Dateisysteme

Dateitypen und Attribute

Zugriffsrechte und Rechteverwaltung

Verzeichnisse

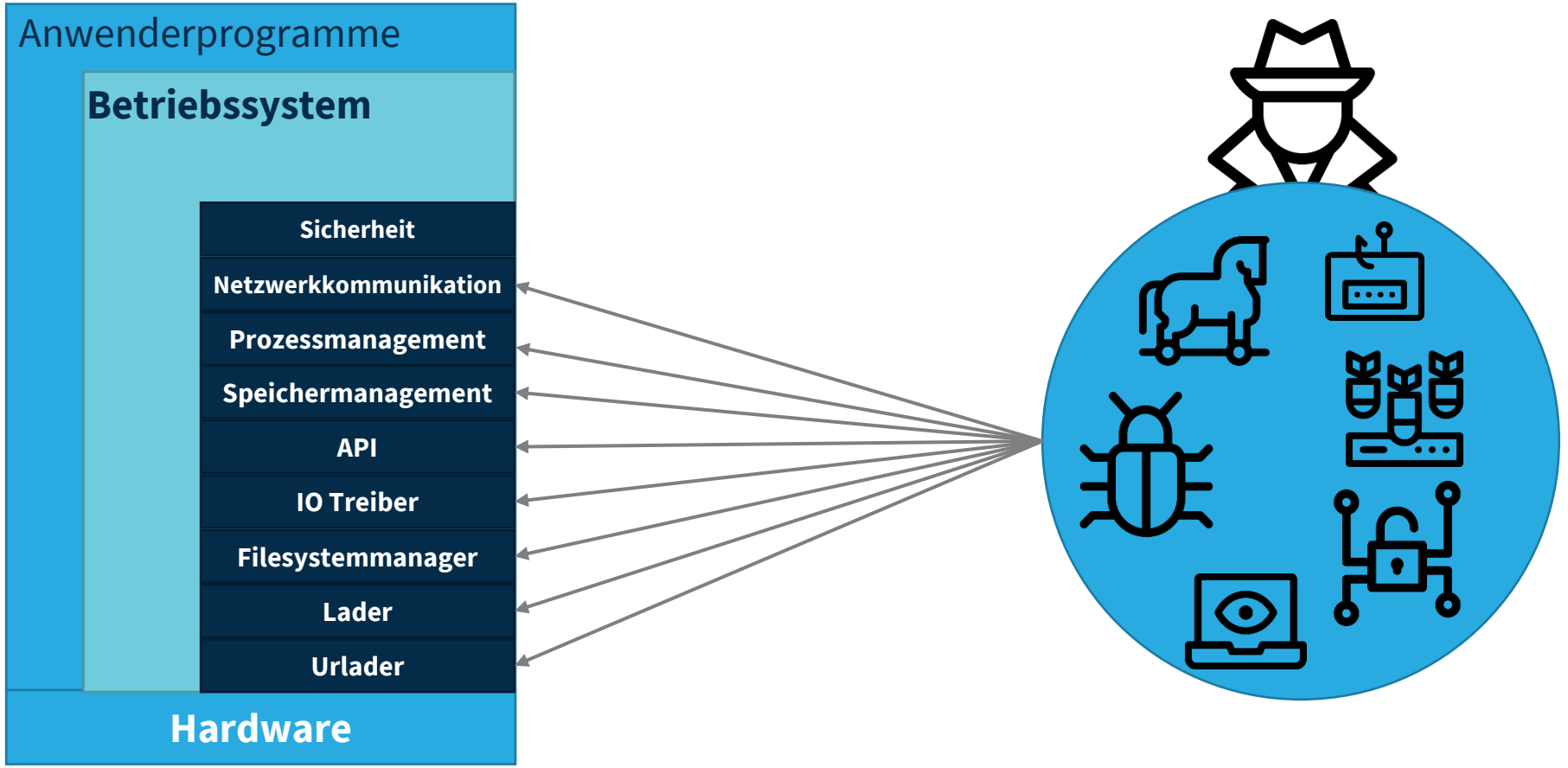
Dateizugriff

Implementierung von Dateisystemen

Realisierung verschiedener Dateisysteme

SECURNITE

Alle Ebenen von Betriebssystemen haben eine Angriffsfläche



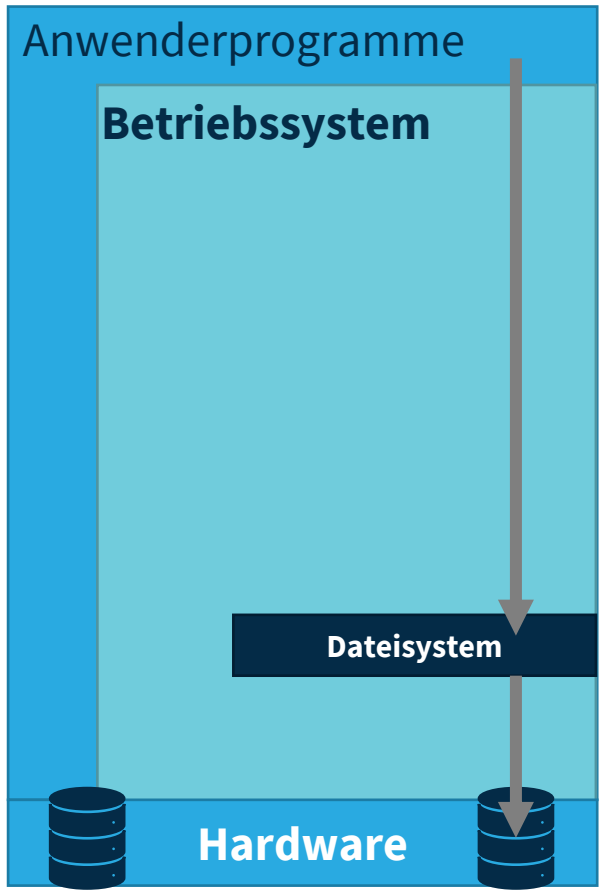
SECURNITE

Einführung in Dateisysteme

Definition: Dateisystem

- Betriebssystemteil, der sich mit **Dateien** befasst
- **Aufgabe** des Betriebssystems:
  - dauerhaftes **Speichern** von Informationen
  - Speicherung von Daten auf Festplatten, USB-Sticks usw.
- Organisationseinheit: Datei
- Dateisystem bezeichnet auch die **Verwaltungsstruktur** einer Partition

Aufgaben



Betriebssystem Aufruf

- Dauerhafte **Speicherung** von Daten in Form benannter Objekte
- Bereiche auf dem **Speichermedium** werden mit einem Dateisystem versehen
- Stellt die **Verwaltungsstruktur** für Objekte bereit
- **Anfrage** nach Inhalt einer Datei muss vom **Betriebssystem** beantwortet werden

Verwaltungsstruktur

- **Hierarchische** Strukturierung: Verzeichnisse mit Objekten

- **Beispiele**:
  - **Reguläre Dateien**: Text-, Binärdateien
  - Verzeichnisse
  - **Gerätedateien**: Schnittstelle zwischen Hardware und Anwendungsprogrammen
  - **Links**: Verweise auf Dateien

Dateibenennung

- Benennung durch **Dateinamen**
- **Zugriff** über Namen
- **Regeln** für die Benennung **variieren** von System zu System, wie auf den folgenden Folien für MS-DOS, Windows und Linux/Unix zu sehen ist

Dateibenennung

- **MS-DOS:** Dateinamen bestanden aus
  - Eigentlichem **Namen** (bis zu acht gültige Zeichen, beginnend mit Buchstaben)
  - **Dateierweiterung** nach Punkt (bis zu drei Zeichen, gibt den Typ an)
  - **Keine** Unterscheidung zwischen **Gross-** und **Kleinschreibung**

Dateibenennung

- **Windows**: Dateien bestehen aus
  - **Name,** bis zu 256 Zeichen (ab 95 bis 7) bzw. 32000 Zeichen (ab Windows 8)
  - Mehrere Punkte möglich, letzter legt **Dateityp** fest
  - Keine Unterscheidung zwischen **Gross-** und **Kleinschreibung**
  - **Nicht erlaubt:** Einzelne Sonderzeichen, reservierte Wörter: z.B. q"uote"s.txt, frage?.txt, aux.txt

Dateibenennung

- **Unix/Linux**: Dateien bestehen aus
  - **Name,** bis zu 255 Zeichen, mehrere Punkte erlaubt
  - Unterscheidung zwischen **Gross-** und **Kleinschreibung**
  - **Dateierweiterung**: nur Konvention zur Übersichtlichkeit
  - Dateien oder Verzeichnisse, deren Name mit einem Punkt beginnt, sind **„versteckte“ Dateien** und werden nur angezeigt, wenn der Benutzer dies explizit angibt (ls -a)

Dateitypen und Attribute

Reguläre Dateien

- **Textdateien**
  - Folge von Zeilen unterschiedlicher Länge
  - Inhalt interpretierbar gemäß **Zeichensatz** (z.B. Unicode, ISO, ASCII)
  - Mit Texteditor editierbar

- **Binärdateien**
  - Interpretation **anwendungsabhängig**
  - Verwendungszweck typischerweise als Dateiendung angegeben

Reguläre Dateien

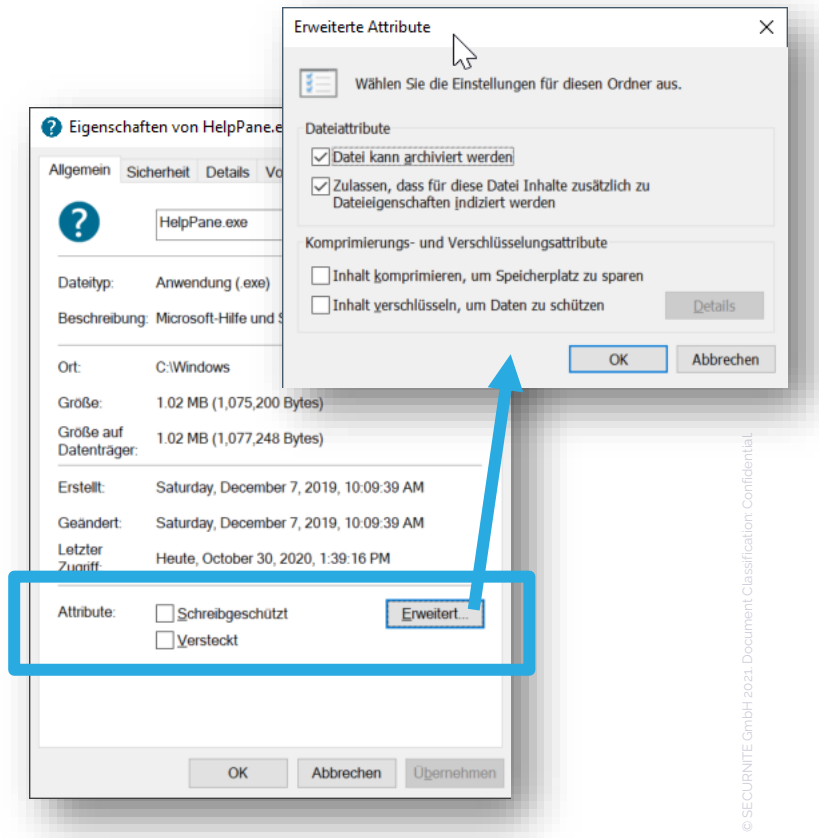
- **Verzeichnisse**
  - **Systemdateien** zur Strukturierung des Dateisystems

- **Gerätedateien**
  - Modellierung von **E/A-Geräten**
    - Spezielle **Zeichendateien** zur Modellierung serieller Geräte, z.B. Drucker, Netzwerke
    - Spezielle **Blockdateien** zur Modellierung von Plattenspeicher, z.B. Festplatten oder Images

Dateiattribute

- **Zusatzinformationen** über Datei, die das Betriebssystem speichert (Metadaten)

- **Beispiele**
  - **Schutz**: Wer kann auf die Datei zugreifen
  - **Passwort**: Passwort für den Zugriff auf die Datei
  - **Urheber**: ID der Person, die die Datei erzeugt hat
  - **Eigentümer/in**: Aktuelle Eigentümer/in
  - **Read-only-Flag**: 0: Lesen/Schreiben; 1: nur Lesen
  - **Hidden-Flag**: 0: normal; 1: in Listen Sichtbar



SECURNITE

Zugriffsrechte und Rechteverwaltung

Zugriffsrechte: Linux

- **Zugriffsarten**
  - Lesender Zugriff
  - Schreibender Zugriff
  - Ausführung

- **Zugreifende**
  - Dateieigentümer/in
  - Benutzergruppe der Dateieigentümer/in
  - Alle anderen Benutzer/innen

Zugriffsrechte: Linux

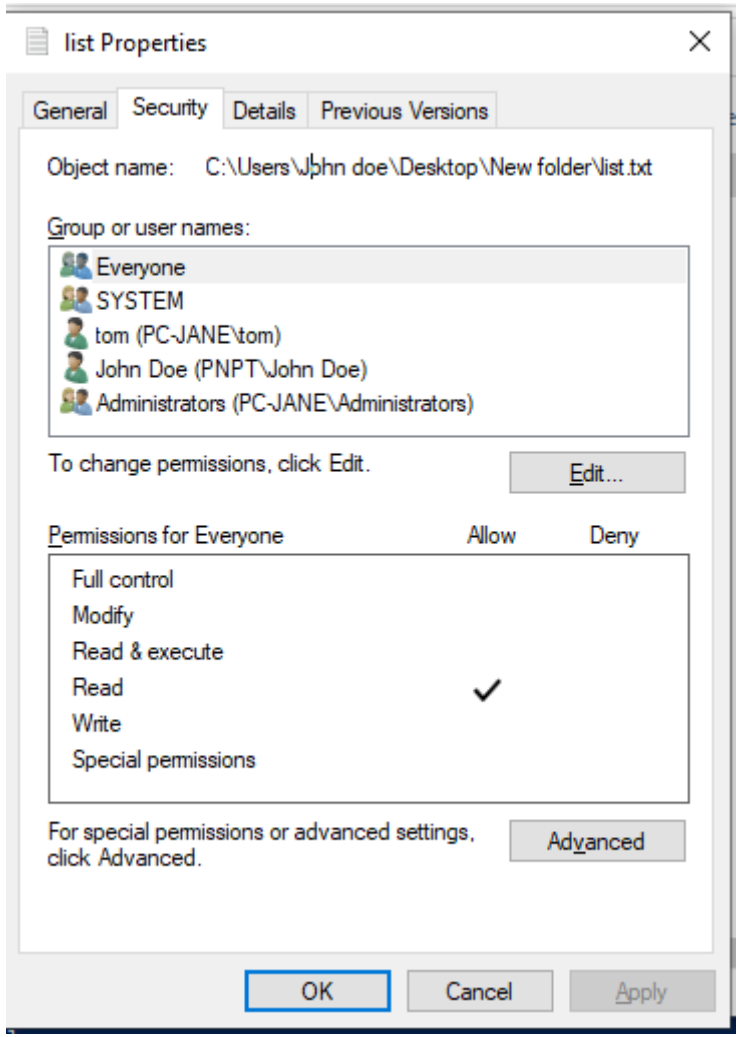
```
$ ls -l
drwxr-xr-x 2 hk1032 uni 52          25. Okt 20:36 folien
drwx------ 3 hk1032 uni 27          25. Sep 20:40 klausur
-rw-r--r-- 1 hk1032 uni 285696      10. Okt 11:42 zeitplan.ods
```

- **Felder**
  - **Typ des Eintrags**: Datei, Verzeichnis
  - **Rechte**: Besitzer, Gruppenbesitzer, alle anderen Benutzer
  - Anzahl Hardlinks (Datei), Anzahl Unterverzeichnisse (Verzeichnis)
  - Besitzer
  - Gruppenbesitzer
  - Speicherplatzverbrauch
  - Datum und Zeit der letzten Modifikation
  - Name

SECURNITE

Zugriffsrechte: Windows

- **Standard Zugriffsarten**
  - Full Control
  - Modify
  - Read & Execute
  - List Folder Contents
  - Read
  - Write



- **Standard Zugreifende**
  - Name des Benutzer
    - Tom
    - John Doe
  - Benutzergruppe
    - Everyone
    - System
    - Administrators

Rechteverwaltung: Linux

- `chmod` verändert **Zugriffsrechte** von Dateien
- Zuerst **Benutzertyp**: u=user, g=group, o=others oder a=all
- Dann entweder
  - **+** um Rechte zu setzen, oder
  - **–** um Rechte zu entziehen, oder
  - **=** um explizit angegebene Rechte zu setzen (und restlichen zu entziehen)
- Am Schluss folgen die **Rechte**, Beispiel: `chmod go-rw dateiname.txt`

SECURNITE

Rechteverwaltung: Linux Standard Rechte

- **Reguläre** Dateien
  - r: Datei darf gelesen werden **(read)**
  - w: Datei darf geändert werden **(write)**
  - x: Datei darf als Programm ausgeführt werden **(execute)**

- **Verzeichnisse**
  - r: Der Inhalt des Verzeichnisses darf gelesen werden
  - w: Der Inhalt des Verzeichnisses darf geändert werden: Dateien anlegen, umbenennen, löschen
  - x: Man darf in das Verzeichnis wechseln und die Objekte darin benutzen

SECURNITE

Rechteverwaltung: Linux Sonder Rechte

- **SUID** (set user ID)
  - Erweitertes **Zugriffsrecht** für Dateien
  - Unprivilegierte Benutzer erlangen **kontrollierten Zugriff** auf privilegierte Dateien
  - Setzen des Bits: `chmod u+s datei`
  - Ausführung mit den Rechten der **Besitzer/in** der Datei
  - Optische **Notation** bei ls -l: `-rwsr-x---`
  - root-User: Hat Zugriff auf alles (Superuser)

Rechteverwaltung: Linux Sonder Rechte

- **SGID** (set group ID)
  - Ausführung mit den Rechten der **Gruppe**, der die Datei/das Verzeichnis gehört (anstatt mit den Rechten der Gruppe, die ausführt)
  - **Verzeichnisse**: Neu angelegte Dateien gehören der Gruppe, der auch das Verzeichnis gehört (anstatt der Gruppe, die eine Datei erzeugt)
  - Setzen des Bits: `chmod g+s verzeichnis`
  - Optische **Notation** bei ls -l: `drwxrws---`

Rechteverwaltung: Linux Sonder Rechte

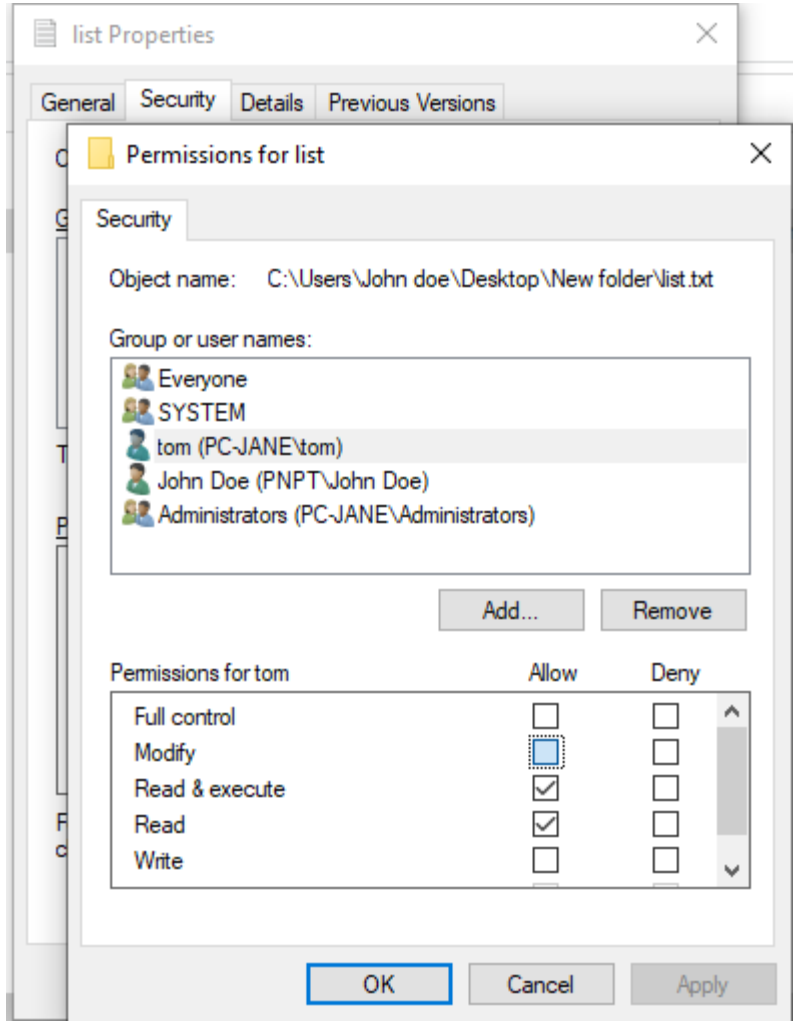
- **SVTX** (save text bit, sticky bit):
  - Zum **Löschen** der Datei in einem Verzeichnis muss der Benutzer/in auch der **Eigentümer/in** der Datei oder des Verzeichnissen sein
  - Optische **Notation**: `drwxrwxrwt`
  - SVTX verhindert, dass jeder alles löschen darf

Rechteverwaltung: Linux Sonder Rechte

- **Access Control Lists** (ACLs)
  - Ermöglichen **ergänzende**, **komplexere** Zugriffsrechte
  - **Einzelnen** Nutzer/innen (oder auch Gruppen) können gezielt Rechte an einzelnen Dateien gegeben bzw. entzogen werden
  - Damit ist z.B. Folgendes möglich
    - Zwei Benutzer/innen haben das Schreibrecht
    - Sonst niemand
  - Optische **Notation**: `-rw-r--r--+`

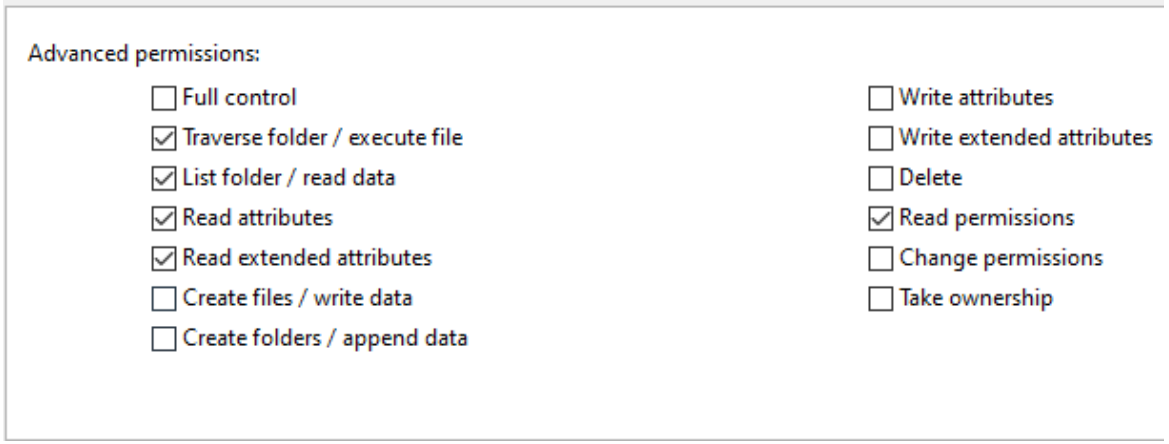
Rechteverwaltung: Windows

- In Windows können Sie die Rechte im Sicherheitsmenü ändern
- Schritt 1: Festlegen des Objekts (Benutzername, Gruppe...) für die Vergabe von Berechtigungen
  - Tom Benutzer
- Schritt 2: Erteilen von Rechten für ein bestimmtes Objekt
  - Berechtigungen zum Ändern erteilen



Rechteverwaltung: Windows Sonder Rechte

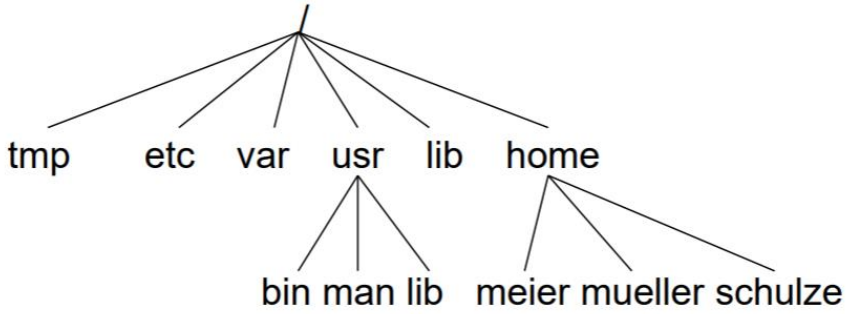
- **Eigentümerschaft übernehmen**: Erlaubt oder verweigert die Übernahme des Eigentums an der Datei oder dem Ordner.
- **Attribute schreiben**: Erlaubt oder verweigert die Änderung der Attribute einer Datei oder eines Ordners, z. B. schreibgeschützt oder versteckt.
- **Berechtigungen ändern:** Erlaubt oder verweigert das Ändern von Berechtigungen für die Datei oder den Ordner, z. B. Lesen, löschen und Schreiben.



SECURNITE

Verzeichnisse

Verzeichnisbaum: Linux




```
user@kali:/$ tree /
/
├── 0
├── bin → usr/bin
├── boot
│   ├── config-5.18.0-kali5-amd64
│   ├── config-5.18.0-kali7-amd64
│   ├── grub
│   │   ├── fonts
│   │   │   └── unicode.pf2
│   │   ├── grub.cfg
│   │   ├── grubenv
│   │   ├── i386-pc
│   │   │   ├── 915resolution.mod
│   │   │   ├── acpi.mod
│   │   │   ├── adler32.mod
│   │   │   ├── affs.mod
```

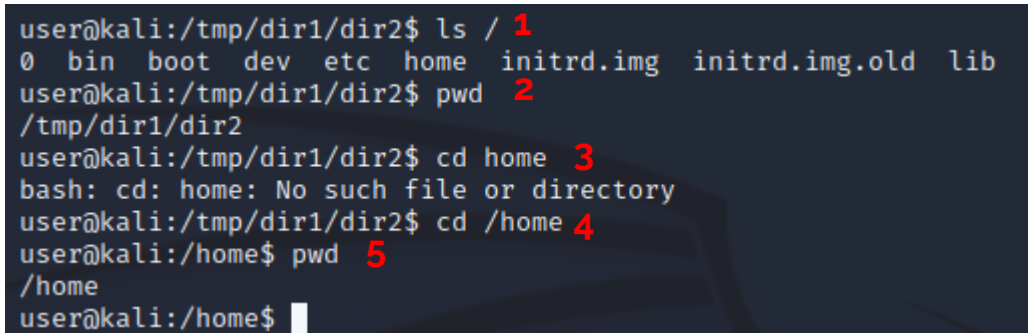
- **Baumartige,** hierarchische Strukturierung
  - **Wurzelverzeichnis** (root directory)
  - Restliche Verzeichnisse baumartig angehängt

Absolute Pfade beginnen mit /

Relative Pfade beziehen sich auf das aktuelle Arbeitsverzeichnis

Verzeichnisbaum: Linux

Navigieren im Verzeichnisbaum



Schritte:

1. Anzeige eines Verzeichnisses durch `ls` („list“), hier die Angabe eines **absoluten Pfads**
2. Anzeige durch `pwd` in welchem Verzeichnis sich der Benutzer befindet
3. Wechsel des aktuellen Arbeitsverzeichnisses durch `cd` („change directory“) mit Angabe eines **relativen Pfads** (ohne Angabe von **/**)
4. Wechsel des aktuellen Arbeitsverzeichnisses mit Angabe eines **absoluten Pfads** (mit Angabe von **/**)
5. Anzeige des aktuellen Verzeichnisses des Benutzers

Verzeichnisbaum: Linux

Zusammenbau

- Temporäres, manuelles **Anhängen** (Mounten) von Dateisystemen:
  - Beispiel: `mount /dev/sda1 /mnt/extern`
  - Hängt Dateisystem in /dev/sda1 an das Verzeichnis /mnt/extern im bestehenden Verzeichnisbaum an
  - /mnt/extern = Mountpoint
  - Evtl. vorhandene Einträge in /mnt/extern werden temporär verdeckt und sind nach Abhängen wieder zugreifbar

Verzeichnisbaum: Linux

Zusammenbau

- Abhängen mit `umount /mnt/extern`

- Geht nur, wenn keine Datei im angehängten Dateisystem **geöffnet** ist

- **Graphische Oberfläche**:
  - Mounten von Wechseldatenträgern (USB-Sticks, CDs, externe Festplatten, etc.) geschieht meist **automatisch**
  - Typischerweise im Verzeichnis /media oder /Volumes

Symbolische Links: Linux

- **Zweck**: Ansprechen desselben Objekts mit mehreren Namen
- Anlegen: `ln -s ziel linkname`
- **Interaktionen**
  - Löschen/Verschieben/Umbenennen von Referenzobjekt: Link zeigt ins Leere
  - Löschen von Link: Referenzobjekt ist unverändert
- **Rechte** am Link: Entsprechend Referenzobjekt

SECURNITE

Harte Links: Linux

- **Anlegen**: `ln ziel linkname`
- Erstellt **Verzeichniseintrag** in Dateisystem mit weiterem **Namen** für Dateiobjekt
- Zulässiges Referenzobjekt: Dateiobjekt in demselben Dateisystem (Festplattenpartition)
- Nach **Umbenennen/Verschieben** funktioniert Link noch

Harte Links: Linux

- Dateiobjekt mit n Hardlinks hat **Linkzähler** n+1
- Löschen eines Links: Dekrementieren des Linkzählers
- Löschen des Dateiobjekts: Dekrementieren des Linkzählers
- Dateiobjekt wird erst dann wirklich gelöscht, wenn Linkzähler den Wert 0 hat

Operationen auf Verzeichnisse

- **Create**: Erzeugen
- **Delete**: Löschen
- **Opendir**: Öffnen
- **Closedir**: Schliessen
- **Readdir**: Nächsten Verzeichniseintrag lesen
- **Get attributes:** Lesen der Verzeichnisattribute
- **Set attributes:** Verändern der Verzeichnisattribute
- **Rename**: Umbenennen des Verzeichnis

SECURNITE

Dateizugriff

Dateizugriff

- **Sequentieller Zugriff** (Folgezugriff)
  - Alle Bytes können nur **nacheinander** vom Datenspeicher gelesen werden
  - Kein **Überspringen** möglich
  - Um auf einen bestimmten Datensatz zugreifen zu können, müssen alle Datensätze zwischen **Start-** und **Zielposition** besucht werden
  - Die Zugriffszeit ist von der **Entfernung** der Datensätze vom ersten Datensatz abhängig

Dateizugriff

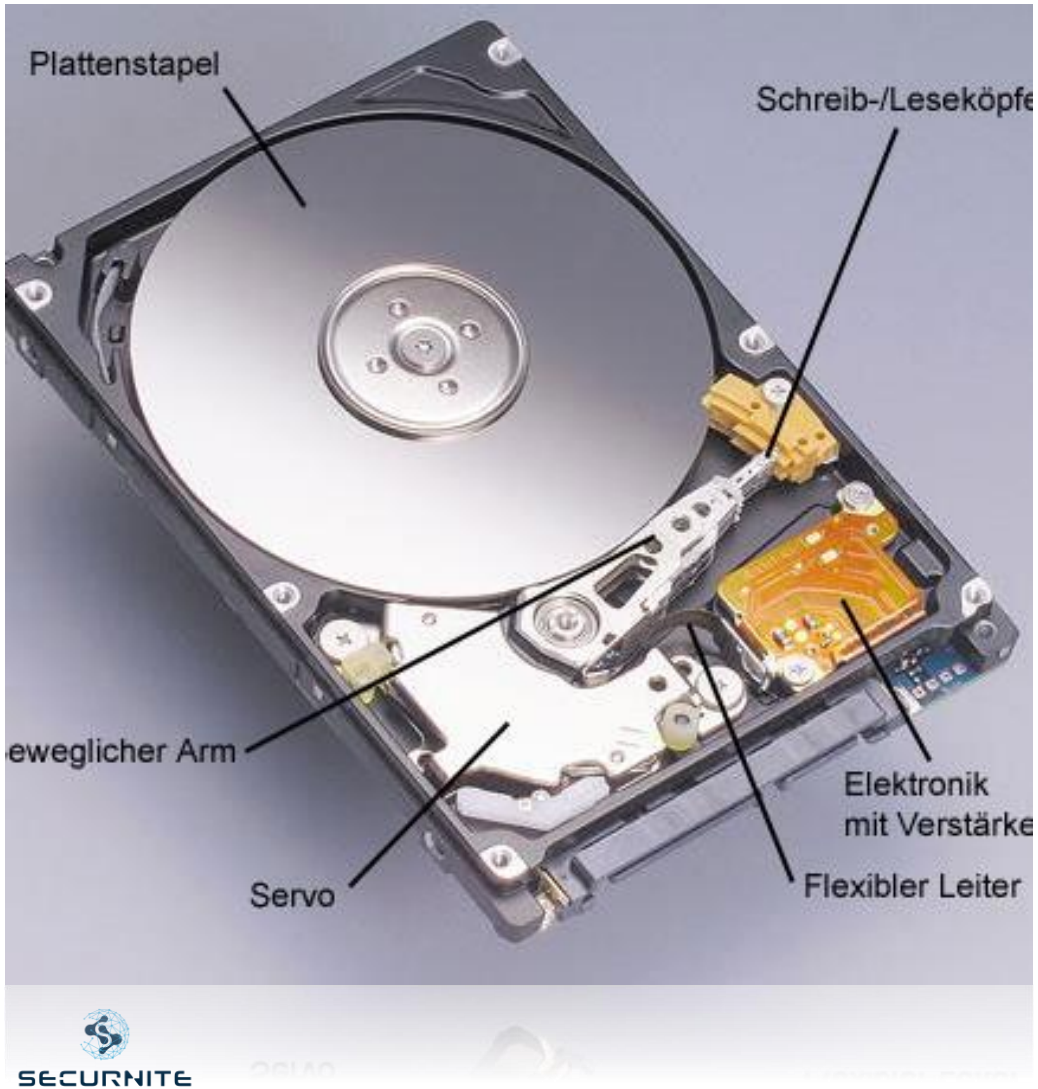
- **Wahlfreier Zugriff** (Direktzugriff, Random Access)
  - Zugriff auf ein **beliebiges** Element in konstanter Zeit
  - Bytes/Datensätze können in beliebiger **Reihenfolge** ausgelesen werden
  - Befehl **seek**: setzt den Dateizeiger auf eine beliebige Stelle (Byteposition innerhalb der Datei)
  - Danach kann die Datei sequentiell von dieser Position gelesen werden

Operationen auf Dateien

- **Create**: Erzeugen
- **Delete**: Löschen
- **Open**: Öffnen
- **Close**: Schließen
- **Read**: Lesen von aktueller Position
- **Write**: Schreiben an aktuelle Position
- **Append**: Anhängen an Dateiende
- **Seek**: Dateizeiger an beliebige Stelle
- **Get Attributes**: Lesen der Dateiattribute
- **Set Attributes**: Verändern der Dateiattribute
- **Rename**: Umbenennen

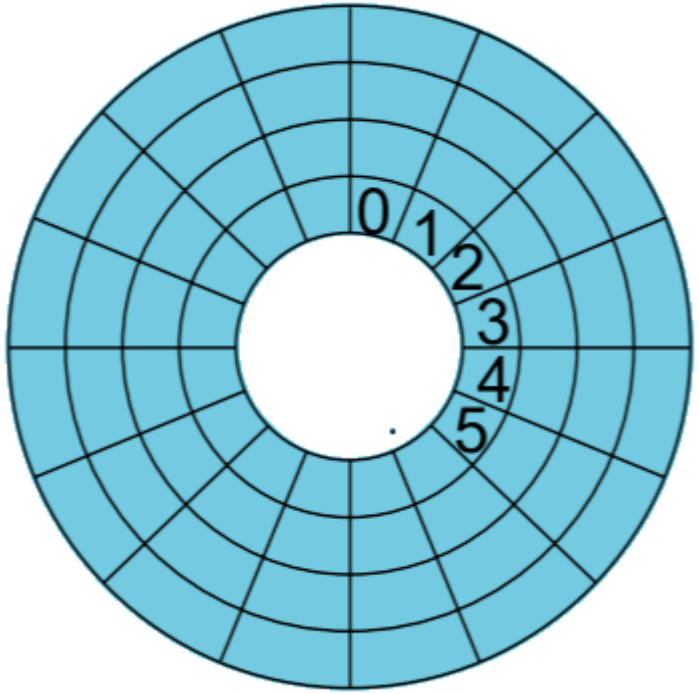
Implementierung von Dateisytemen

Wiederholung: Festplatten



- Eine Festplatte besteht aus mehreren **Scheiben** und einem **Lesekopf**
- Festplatten können in eine oder mehrere **Partitionen** unterteilt werden
- Einzelne Partitionen können unabhängige **Dateisysteme** besitzen (z.B. Windows und Linux Dateisystem)

Wiederholung: Festplatten

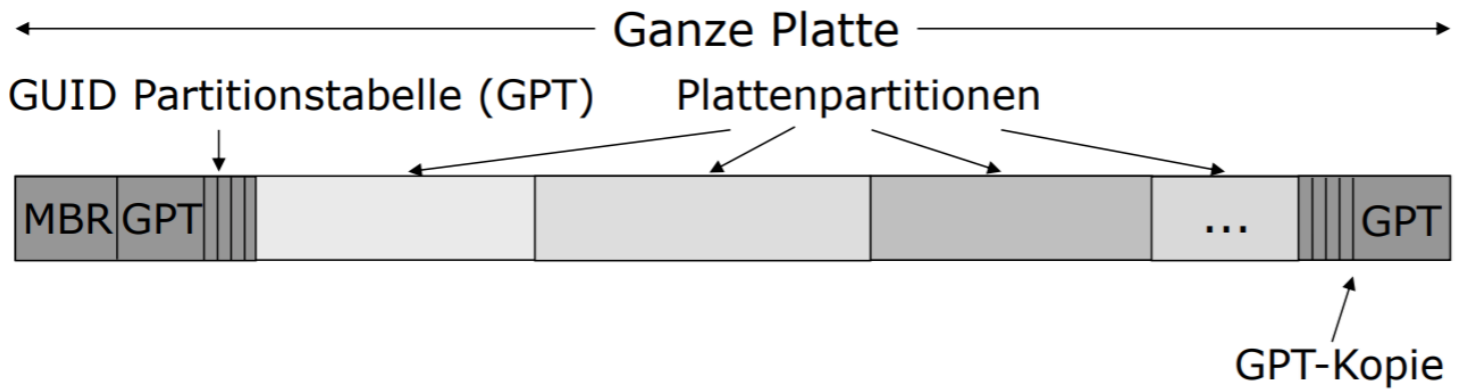


- Scheiben sind eingeteilt in **Blöcke** (Sektoren)
- alle Blöcke der Festplatte sind **durchnummeriert**

Wiederholung: Festplatten

Layout

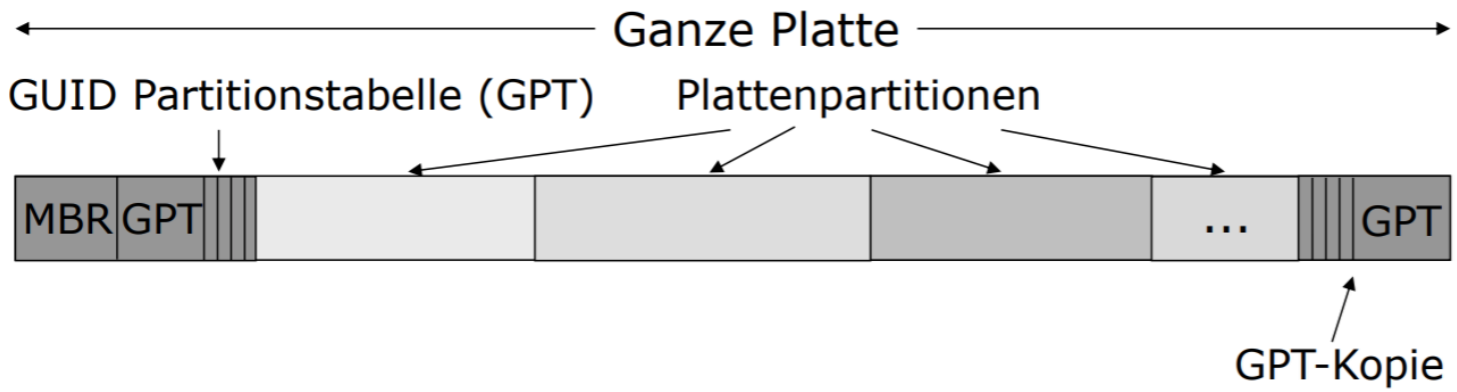
- **GPT-formatierte Festplatte**
  - GPT = GUID (Globally Unique Identifier) Partitionstabelle
  - GPT enthält Anfangs- und Endadresse jeder **Partition**
  - Bei alten Betriebssystemen: Erster Sektor der Platte enthält **MBR** (Master Boot Record)



Wiederholung: Festplatten

Layout

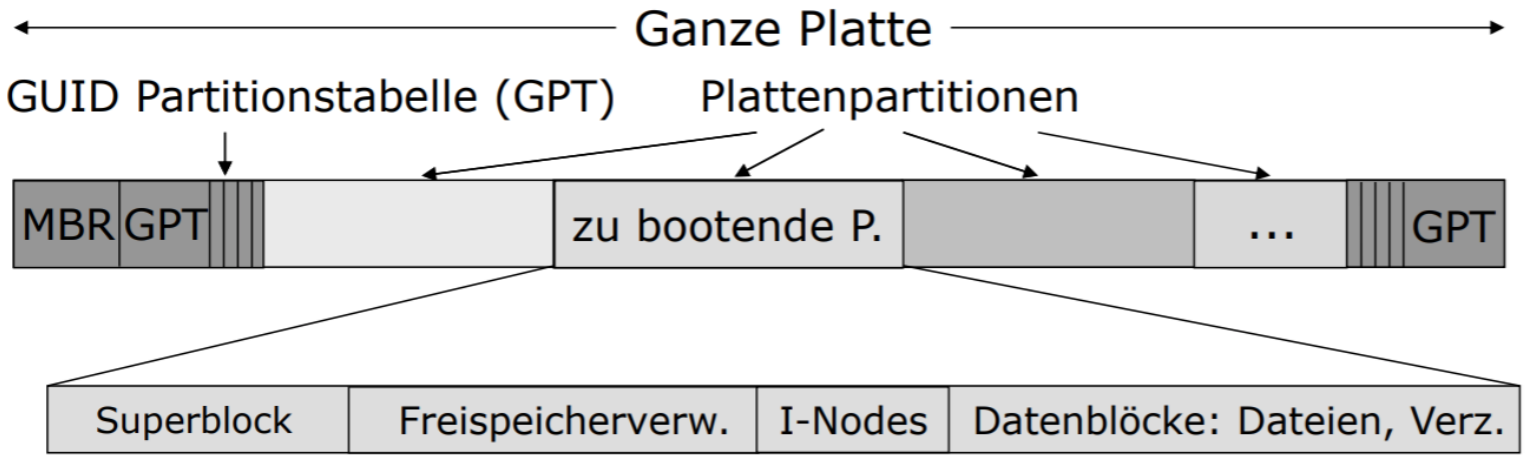
- Wahl der zu bootenden Partition durch **Boot Manager** (vgl. OSSEC-01/03)
- **Betriebssystem** von der gewählten Partition wird eingelesen und gestartet



SECURNITE

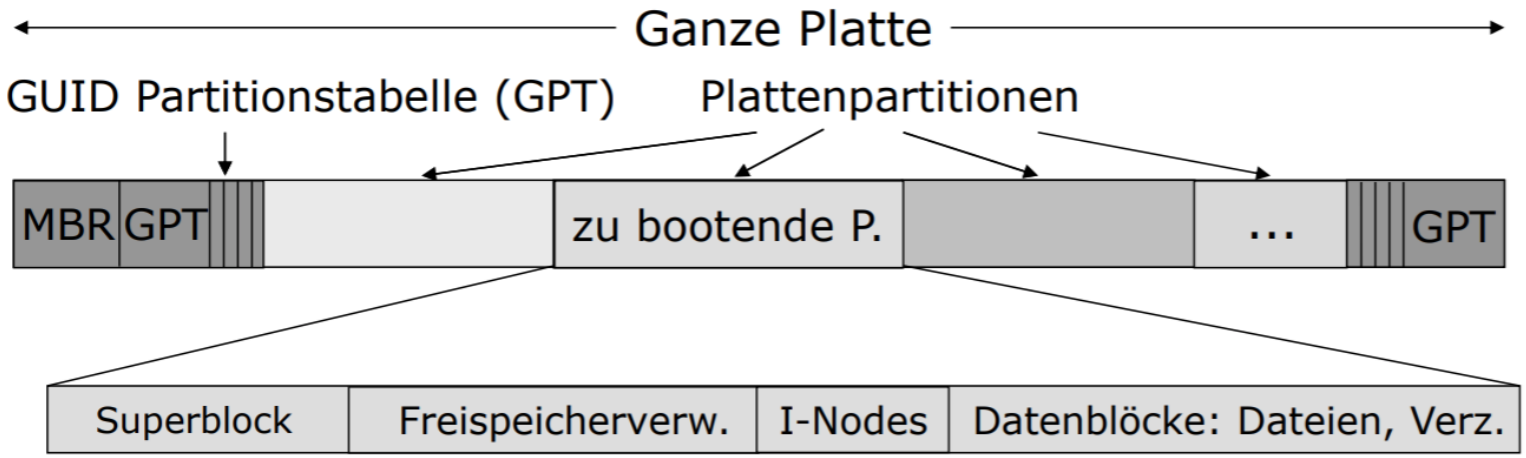
Festplatten Layout: Linux

- **Superblock** enthält Schlüsselparameter des **Dateisystems** (z.B. Name des Dateisystemtyps, Anzahl Blöcke)
- **Freispeicherverwaltung**: Informationen über freie Blöcke im Dateisystem



Festplatten Layout: Linux

- **I-Nodes** (Index Node): Enthalten Metadaten der Dateien (Eigentümer, Zugriffsrechte, Dateityp, Größe, Linkzähler, etc.)
- Zu jeder **Datei** gehört ein **I-Node**
- **Datenblöcke**: Eigentliche Inhalte der Dateien

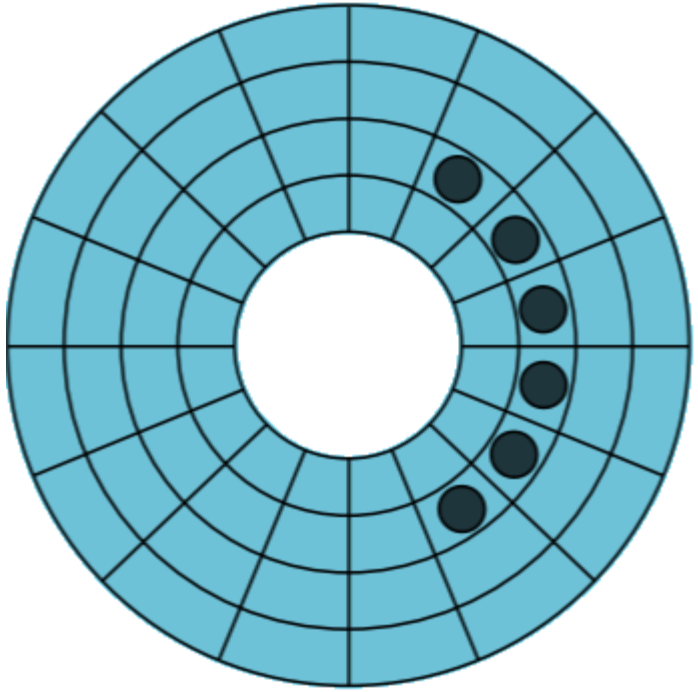


Realisierung verschiedener Dateisysteme

Möglichkeiten zur Realisierung von Dateien

- Drei verschiedene **Alternativen** zur Realisierung von Dateien
  - **Zusammenhängende** Belegung
  - Belegung durch **verkettete Listen**
  - **I-Nodes** (bzw. File Records)

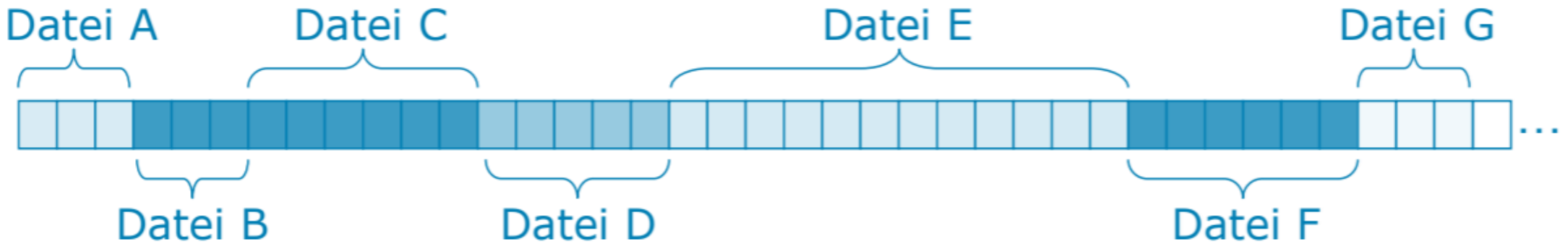
Zusammenhängende Blöcke



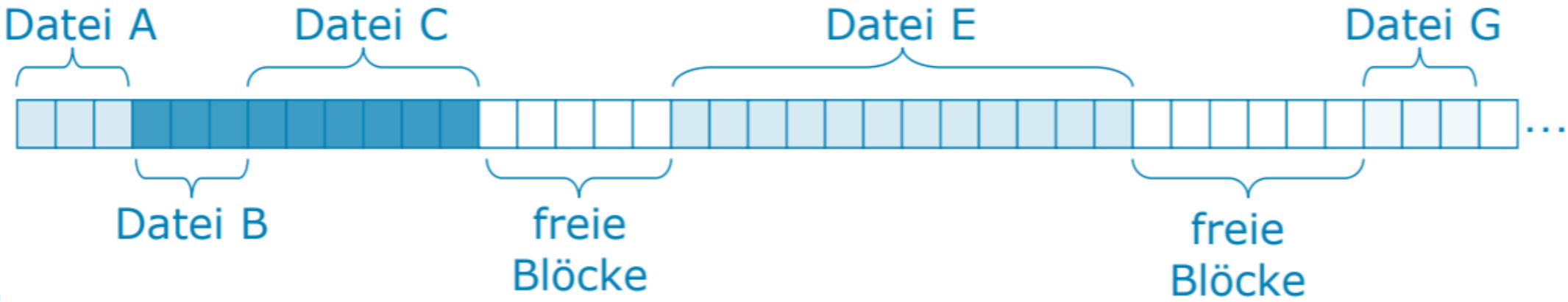
- Abspeicherung von Dateien durch zusammenhängende Menge von **Plattenblöcken**
- **Vorteil**: Lesegeschwindigkeit (wenige Leseoperationen für gesamte Datei)
- **Nachteil**: Externe Fragmentierung der Platte
- Verschiebung der Blöcke (Defragmentierung) ist ineffizient

Zusammenhängende Blöcke

Fragmentierung



- **Löschen** von Datei D und F



SECURNITE

Zusammenhängende Blöcke

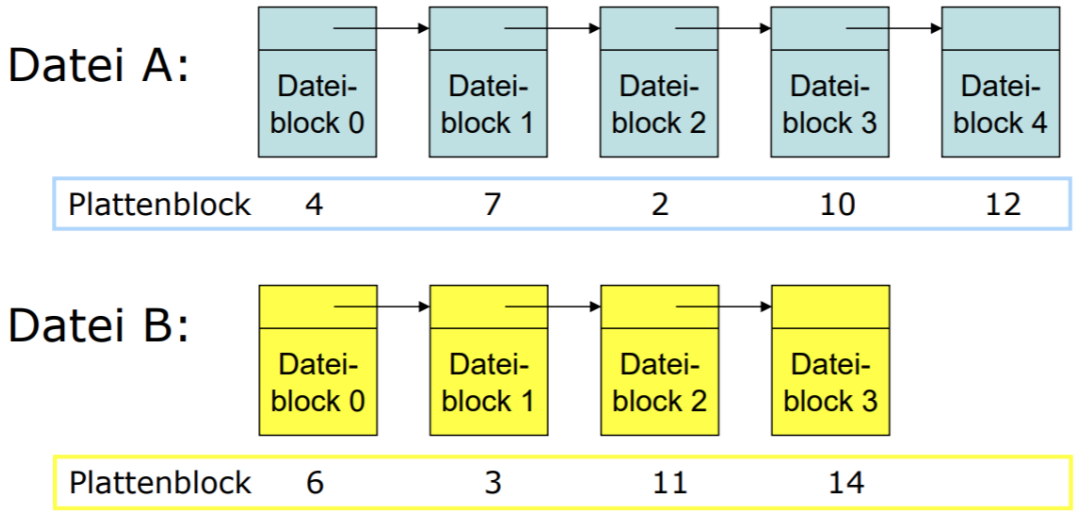
- Verwaltung der entstehenden **Lücken**
  - in **Listen** möglich
  - Endgrösse von Dateien muss bekannt sein, um passende Lücke zu finden
  - Dateien können **nicht wachsen**!

- **Anwendung**: Dateisysteme für **einmal beschreibbare** Medien z.B. DVD, da Dateigrösse im Voraus bekannt

Verkettete Listen

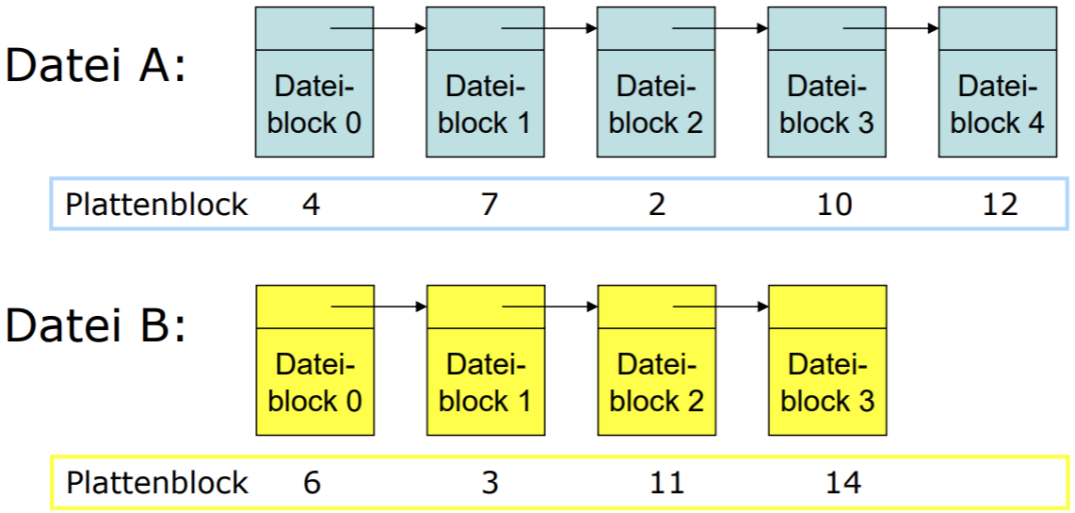
Realisierung von Zeigern auf den nächsten Block

- Dateien gespeichert als verkettete Listen von Plattenblöcken



Verkettete Listen

Realisierung von Zeigern auf den nächsten Block

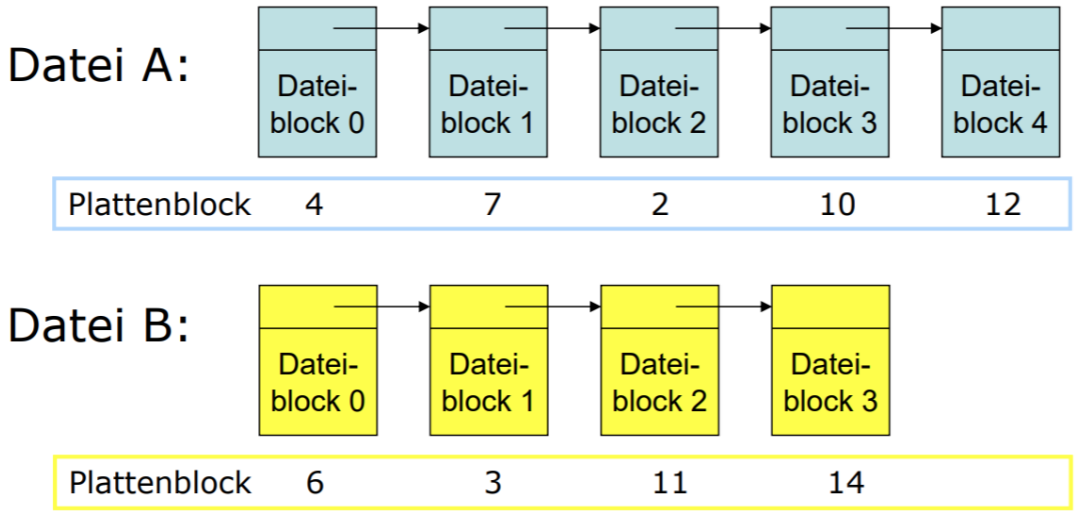


Variante 1

- **Vorteile**
  - Fragmentierung der Festplatte führt nicht zu Verlust von Speicherplatz
  - Dateien beliebiger Grösse können angelegt werden
- **Nachteile**
  - Langsamer wahlfreier Zugriff
  - n-1 Lesezugriffe auf die Platte, um Block n zu lokalisieren

Verkettete Listen

Realisierung von Zeigern auf den nächsten Block

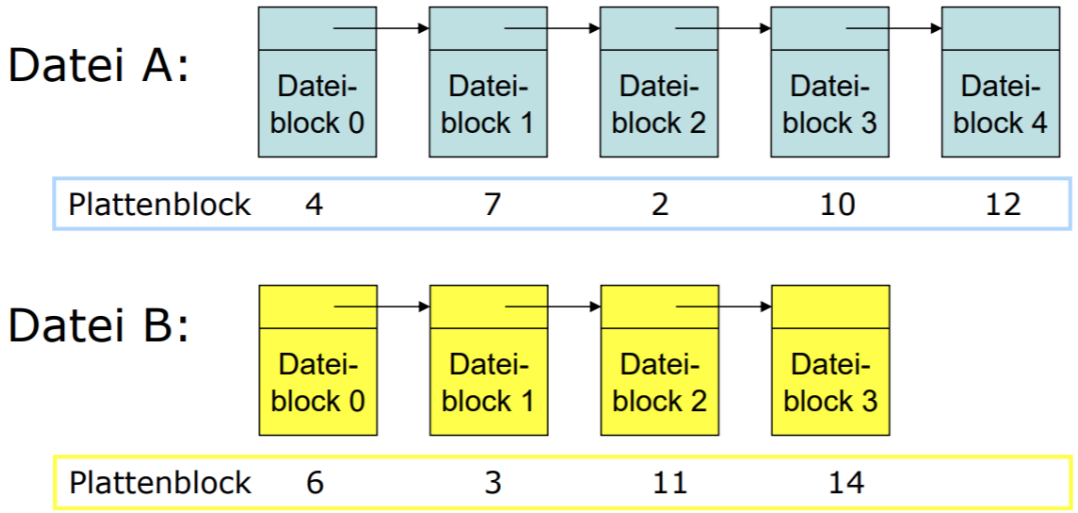


Variante 2

- Information über Verkettung der Blöcke im **Hauptspeicher**
- Ersetze bei **wahlfreiem Zugriff** Plattenzugriffe durch schnellere **Hauptspeicherzugriffe**
- Datei-Allokationstabelle (**FAT=File Allocation Table**) im Hauptspeicher
- Z.B.: FAT12, FAT-16, FAT-32

Verkettete Listen

Realisierung von Zeigern auf den nächsten Block



Variante 2

- **Vorteile**
  - FAT ist im Hauptspeicher
  - Bei Zugriff auf Block n muss eine Kette im HS verfolgt werden
- **Nachteile**
  - Größe der FAT im Speicher: Jeder Block hat einen Zeiger in der FAT
  - Anzahl der Einträge = Gesamtzahl der Plattenblöcke

FAT 32

- **Vorteile**
  - Maximale Partitionsgröße: 2 TiB
  - Die meisten Betriebssysteme können darauf zugreifen
  - Viele externe Geräte verwenden es heute (Digitalkamera, MP3-Player, ...)

- **Nachteile**
  - Größe der FAT
  - Maximale Dateigröße: 4 GiB
  - Grund: 4 Byte großes Feld für die Dateigröße in der Directory-Tabelle

I-Node

- Index-Knoten ist eine **Datenstruktur** zu jeder Datei
- Enthält **Metadaten** und Adressen von Plattenblöcken
- Ermöglicht **Zugriff auf alle Blöcke** der Datei
- Muss nur dann im **Hauptspeicher** sein, wenn die Datei offen ist
- Wenn k Dateien offen sind und ein I-Node n Bytes benötigt: k*n Byte an Speicher werden benötigt

SECURNITE

I-Node

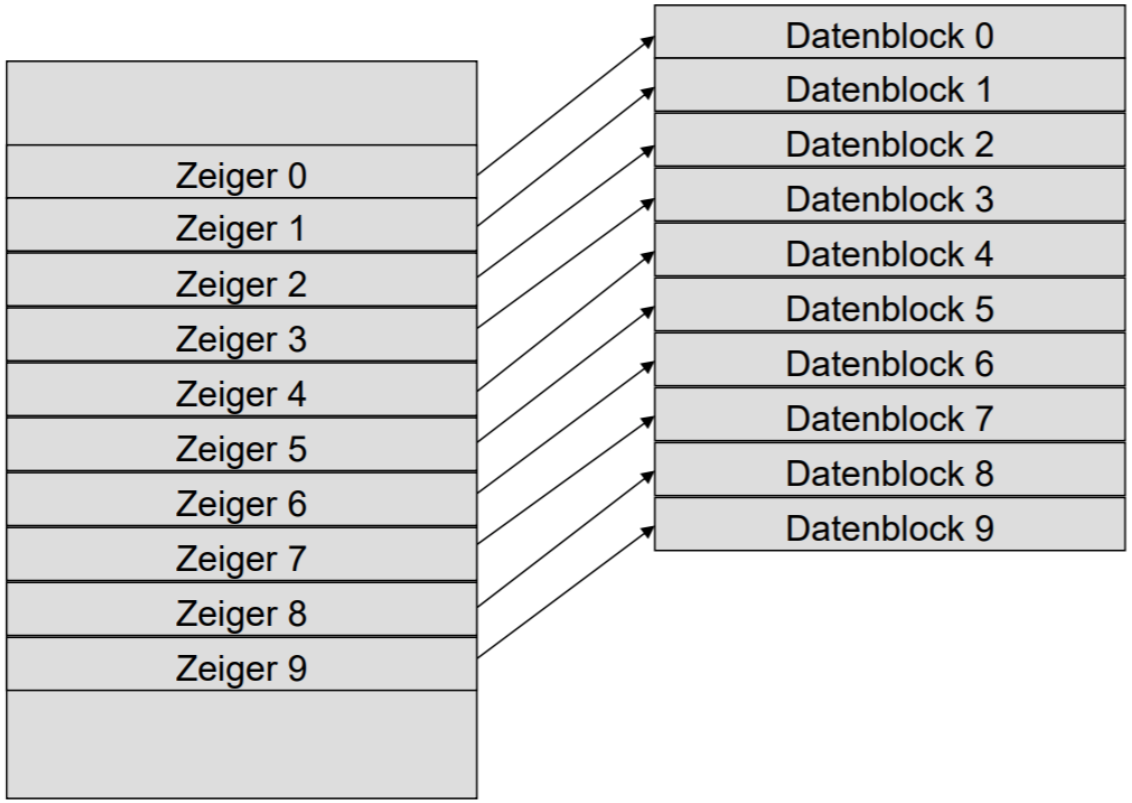
Tabelle

- **I-Node-Tabelle** enthält alle I-Nodes mit Speicheradresse
- Grösse wird **beim Anlegen** des Dateisystems festgelegt
- Ausreichende Anzahl von I-Nodes muss eingeplant werden: Jeder Datei ist ein eindeutiger I-Node zugeordnet
- Z.B. genutzt von **ext4**
  - Blockgröße 4 KiByte
  - Max. Dateisystemgröße 1 EiB = $1{,}153 \cdot 10^{18}$ Byte

SECURNITE

I-Node

Direkte Zeiger

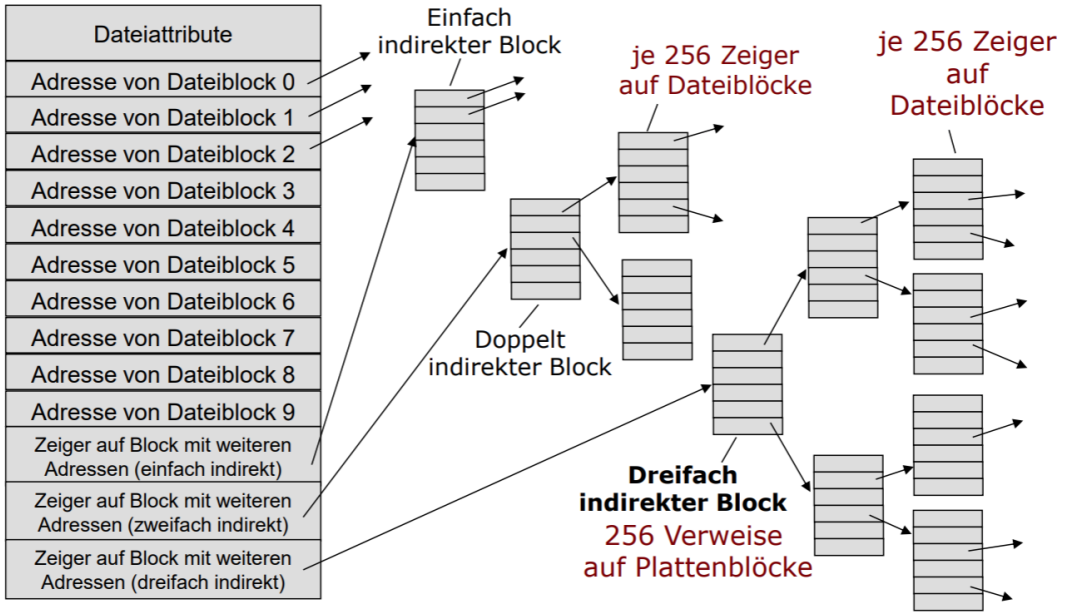


- Viel weniger Bedarf an **Hauptspeicher** als bei FAT System
- Grösse des benötigten **Speichers** proportional zur maximalen Anzahl **gleichzeitig geöffneter Dateien**
- Unabhängig davon, wie gross die Platte ist
- Inhalte Dateien bis 10 KiB können mit **direkten Zeigern** angesprochen werden

SECURNITE

I-Node

Indirekte Zeiger



Für **grössere Dateien** werden Datenblöcke zur Speicherung von weiteren Plattenadressen genutzt

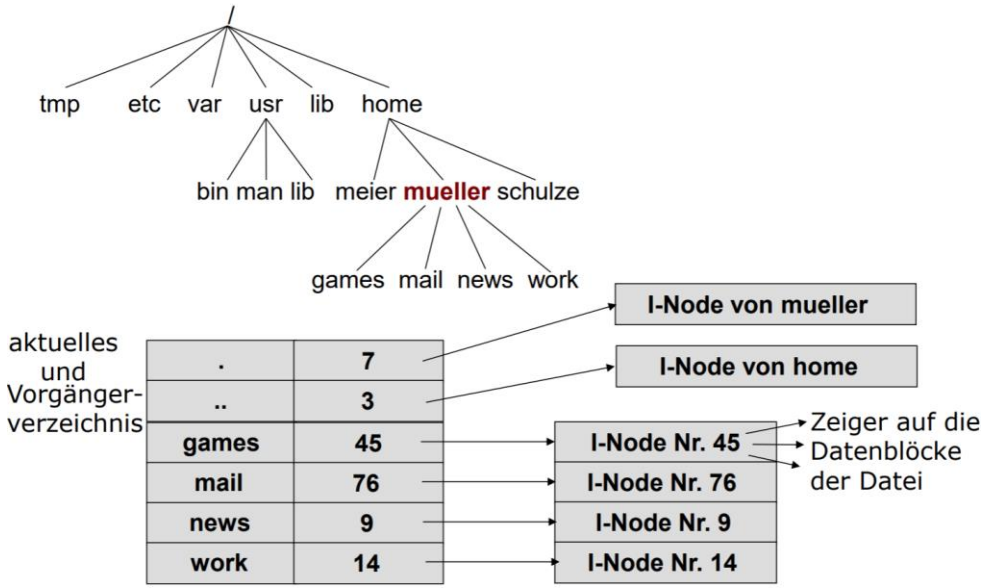
Einfach **indirekter Zeiger** verweist auf Plattenblock, der maximal 1 KiB/4 Byte Zeiger verwalten kann

Zweifach **indirekter Zeiger**: 1 KiB * 256 * 256 = 64 MiB Daten

Dreifach **indirekter Zeiger**: 1 KiB * 256 * 256 * 256 = 16 GiB Daten

I-Node

Verzeichnisse



- **Verzeichnisse** sind ebenfalls Dateien
- **Typ-Feld** kennzeichnet Verzeichnis
- Verzeichnis liefert eine **Abbildung** von Datei- bzw. Verzeichnisnamen auf I-Node-Nummern
- Jeder Verzeichniseintrag ist ein Paar aus **Name** und **I-Node-Nummer**
- Über I-Nodes kommt man zu Inhalten

I-Node

- **Vorteile**
  - Grösse fix und relativ klein
  - kann lange im Hauptspeicher bleiben
  - Auf kleinere Dateien kann direkt zugegriffen werden

- **Nachteile**
  - Mehrere Blöcke müssen bei **langen Dateien** geladen werden

Journaling Filesysteme: NTFS

- Datei = einfache, **unstrukturierte** Folge von Bytes
- **Beliebiger** Inhalt; für das Betriebssystem **transparent**
- **Dynamisch** erweiterbar
- Rechte **verknüpft** mit NT-Benutzern und –Gruppen
- Kann automatisch **komprimiert** abgespeichert werden

Journaling Filesysteme: NTFS

- bis zu **16 Exabytes** lang lt. NTFS-Standard, 16 Terabytes in aktuellen Implementierungen
- **Hard Links:** mehrere Einträge derselben Datei in verschiedenen Verzeichnissen
- **Volume** (Partition) wird durch einen Laufwerksbuchstaben dargestellt z.B. C:
- **Journaling** Filesystem
  - Änderungen an MFT und Dateien werden **protokolliert**
  - Konsistenz der Daten und Metadaten kann wieder hergestellt werden

Journaling Filesysteme: NTFS

- Zusätzlich zum Schreiben der Daten und Metadaten (z.B. Inodes) wird ein **Protokoll** der **Änderungen** geführt

- Alle Änderungen treten als Teil von **Transaktionen** auf, Beispiele:
  - Erzeugen, Löschen, Erweitern, Verkürzen von Dateien
  - Dateiattribute verändern
  - Datei umbenennen

Journaling Filesysteme: NTFS

- Beim **Bootvorgang** wird überprüft, ob die protokollierten Änderungen vorhanden sind:
  - **Redo**: Transaktion kann wiederholt bzw. abgeschlossen werden, falls alle Log-Einträge vorhanden.
  - **Undo**: Angefangene aber nicht beendete Transaktionen werden rückgängig gemacht

- Log-Eintrag wird immer **vor** der eigentlichen Änderung auf Platte geschrieben

- Wurde etwas auf der Platte geändert, steht auch der **Protokolleintrag** dazu auf der Platte

NTFS

- **Vorteile**
  - Grosse Dateien möglich
  - Verschlüsselung
    - Encrypting File System (EFS)
    - Transparent für Anwender
    - DESX, Triple DES, AES
  - Sicherhungsmechanismen
  - Datenkomprimierung

- **Nachteile**
  - Ineffizienter als FAT, da Log File geschrieben werden muss

Gruppenübung

20 Minuten

- Recherchieren Sie die **Hintergründe** zu NTFS Schwachstellen
  - **Gruppe 1:** CVE-2020-0838
  - **Gruppe 2:** CVE-2018-1036

- Präsentieren Sie Ihre **Erkenntnisse (max. 5 Minuten)**. Gehen Sie besonders auf folgende **Punkte** ein
  - Welche **Schwachstelle** wurde ausgenutzt
  - Welche **Möglichkeiten** ergeben sich dadurch für Angreifer
  - Welche **Massnahmen** können ergriffen werden

Übung

30 Min

- Öffnen Sie das folgende Online-Labor und führen Sie die Schritte aus.
  - https://lab.redhat.com/tracks/using-file-permissions