Technische Hochschule Ingolstadt

Fakultät Informatik

Cloud Applications und
Security Engineering (M.Sc)

Studienarbeit

Computer Forensik und Vorfallsbehandlung

Automatisierung forensischer Analysen

Autor

Mario Fuchs
00117827

Prüfer

Prof. Dr. Stefan Hahndel

Abgabedatum

13.06.2024

Selbstständigkeitserklärung nach § 30 Abs. 4 Nr. 7 APO THI

Ich erkläre hiermit, dass ich die Arbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ingolstadt, 24. Juli 2024

Mario Fuchs

Inhaltsverzeichnis

Kapitel 1

Einleitung

Brian Carrier, Gründer von Autopsy und The Sleuth Kit, definiert in seinem weit zitierten Buch *File System Forensic Analysis* [1] eine Reihe relevanter Begriffe im Kontext der Computerforensik. Darunter in Kapitel 1 auf Seite 13 den Begriff *digital forensic investigation* als den Prozess mittels Wissenschaft und Technologie digitale Objekte zu analysieren und darauf basierend Theorien zu entwickeln, die beispielsweise vor Gericht vorgebracht werden können, um Antworten zu aufgetretenen Ereignissen zu liefern.
Eine technische Maßnahme, die in der Informatik häufig verwendet wird, ist die Automatisierung. Dieser Begriff wird unter anderem im Cambridge Wörterburch [2] als den Einsatz von Maschinen und Computern, die ohne menschliche Kontrolle arbeiten können, oder spezifisch im Geschäftskontext als, den Einsatz von Maschinen oder Computern anstelle von Menschen zur Erledigung einer Arbeit, insbesondere in einer Fabrik oder einem Büro, definiert.
Hayes und Kyobe beschreiben in *The Adoption of Automation in Cyber Forensics* [3], dass die Automatisisierung in der Computerforensik oft mit dem Begriff ,,Pust-Button Forensics" (PBF) assoziiert wird. Dieser Ansatz ermöglicht es den Ermittlern, vermeintlich komplexe Analysefunktionen bei Untersuchungen zu verwenden. Insgesamt verspricht man sich dadurch positive Auswirkungen auf Kosten und Effizienz, wird allerdings auch stark kritisiert. Dabei befürchten Forensiker, dass die übermäßige Abhängigkeit von automatisierten Tools zu einem Rückgang des Fachwissens führt und die Qualität der Analysen beeinträchtigt, sowie wichtige Beweise übersehen werden könnten.
Wirft man einen Blick auf den Bereich der Softwareentwicklung, so ist der Einsatz von Automatisierung mittlerweile als bewährter Standard zu betrachten, wie in den jährlich groß durchgeführten Umfragen von Stack Overflow dargestellt. So auch im letzten Jahr [4], gegenüber dem Vorjahr [5] in allen Kategorien ein prozentualer Zuwachs festgestellt werden konnte.
Ebenfalls hat die Automatisierung das Potenzial in der Forensik, die Effizienz von Untersuchen zu steigern, insbesondere in der Vorbereitung und der Übernahme von Routineaufgaben, um so den Ermittlern mehr Zeit für komplexere Analysen zu überlassen.

Kapitel 2

Ziel der Arbeit

Ziel dieser Arbeit ist es, die Vereinbarkeit von Automatisierungstechniken in digitalen forensischen Analysen auf Disk Images zu untersuchen. Um die in Kapitel 1 angesprochene Bedenken zu berücksichtigen, fokussieren sich die entsprechenden Analysen konkret auf Dateisystemanalyse, Datenwiederherstellung sowie darauf aufbauende Datenanalyse und abschließende Aufbereitung und Berichterstattung.

Konkret soll eine Anwendung entwickelt werden, die forensischen Ermitteln wiederkehrende Routineaufgaben abnimmt und die resultierenden Ergebnisse nach Anwendung einen Ersteindruck des zu untersuchenden Objekts vermittelt. Darüber hinaus sollen die aufbereiteten Informationen zur Unterstützung weiterer tieferer Analysen dienen, die schließlich in der Regel manuell von einer Person durchgeführt werden.

Das entwickelte Konzept ist auf GitHub vorzufinden und kann über den folgenden Link `https://github.com/foxx08/forensics-automation` abgerufen werden. Die praktische Umsetzung basiert auf den Werkzeugen und Ansätzen, welche in Kapitel 3 definiert sind. Die eingesetzten Automatisierungstechniken umfassen das Skripting sowie den Einsatz eines Automatisierungswerkzeugs als Bindeglied zur Gewährleistung der sequenziellen Ausführung der Skripte.

Insgesamt soll die Arbeit zeigen, dass durch die Integration von Automatisierungstechniken im richtigen Maße die Effizienz in der forensischen Analysen gesteigert werden kann, ohne dabei Qualität einbüßen zu müssen.

Kapitel 3

Hintergrund

Dieses Kapitel definiert die grundlegenden Konzepte, welche für Kapitel 4 relevant sind und stellt die konkrete Vorgehensweise vor.

3.1 Ablauf der forensischen Untersuchung

Basierend auf der Definition von Brian Carrier, vorgestellt im Kapitel 1, definiert der Autor weiter ab Seite 15 in [1] generelle Richtlinien, welche während einer Untersuchung zwingend eingehalten werden sollten. Diese lauten wie folgt:

- Preservation/Erhaltung: Daten dürfen nicht verändert werden
- Isolation/Isolierung: Daten von der Außenwelt abschirmen
- Correlation/Zusammenhang: Gefälschte Daten identifizieren
- Logging/Protokollierung: Handlungen und Ergebnisse dokumentieren

Die forensische Untersuchung in dieser Arbeit, dargestellt als Flowchart auf Abbildung 3.1, zielt darauf, das erhaltene Testobjekt, beschrieben in Abschnitt 3.1.1, zu analysieren. Dabei sollen zunächst Informationen über die einzelnen Partitionen und deren Dateisystem gesammelt werden. Anschließend sollen sowohl gelöschte als auch aktive Daten für weitere Analyse wiederhergestellt werden.

3.1.1 Testobjekt: USB Image

Zur praktischen Anwendung des in dieser Arbeit vorgestellten Automatisierungskonzepts wurde ein USB-Test-Image, zu Deutsch Datenträgerabbild, der Größe 1 GB verwendet. Das Test-Image wurde freundlicherweise von Professor Hahndel im Zuge der Bearbeitung zur Verfügung gestellt. Das Image hat die Dateiendung `.dd`. Der initiale Hashwert des Abbilds wurde per Mail übermittelt und beträgt `2e8a6d70fe99fbf44f7b38e7355fd29a`.

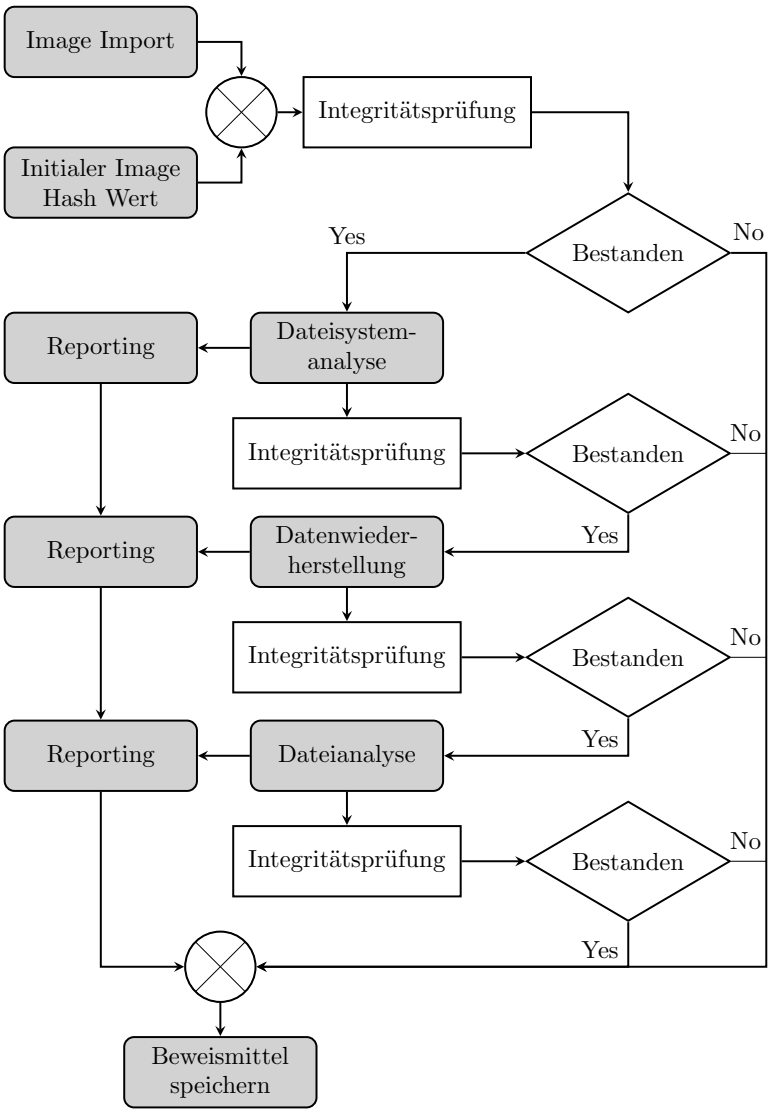

Abbildung 3.1: Forensische Untersuchung des Testobjekts

Dieser Wert dient als Referenzwert und muss während der Analysen im Zuge von Integritätsprüfungen stets unverändert bleiben, um nachzuweisen, dass das Image nicht verändert worden ist.

3.2 Technischer Ansatz

Abbildung 3.2 beschreibt die verwendeten Technologien zur Umsetzung von automatisierten forensischen Analysen. Die Komponente `Python` übernimmt dabei die direkte Umsetzung durch entsprechende Module, die im Abschnitt 3.2.1 näher beschrieben werden. `Jenkins` fungiert als Bindeglied, führt dabei die Pythonskripte in einer bestimmten Reihenfolge aus und speichert die Ergebnisse ab. Als weiteres Tool zur Datenwiederherstellung, insbesondere für gelöschte Dateien, wird `foremost` verwendet, wobei die Wie-

derherstellung der Dateien mittels Data Carving basierend auf der Auswertung von Kopf- und Fußzeilen sowie interner Datenstrukturen erfolgt [6].

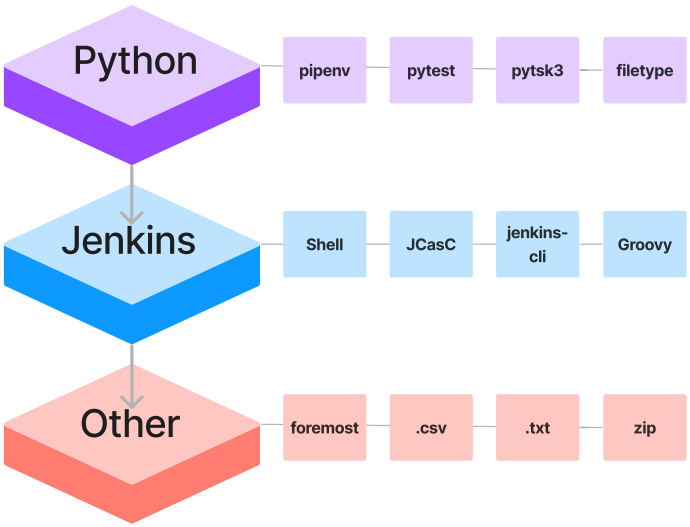

Abbildung 3.2: Tech Stack

3.2.1 Skriptsprache: Python

Python [7] gehört mit zu den beliebtesten Programmiersprachen, wie Stack Overflow in ihrer jährlich durchgeführten Umfrage aus dem Jahr 2023 [8] erneut bestätigt hat. Ebenfalls in der IT-Sicherheit, insbesondere dem ethischen Hacking, scheint Python ebenfalls sich größter Beliebtheit zu erfreuen, wie unter anderem eine Umfrage aus dem Jahr 2022 [9], durchgeführt im German Chaos Computer Club, bestätigt. Python zeichnet sich vor allem dadurch aus, dass neben den Standardbibliotheken ebenfalls eine Vielzahl an Drittanbietermodulen unterstützt und angeboten werden.

3.2.2 Module

Module sind Dateien, die Definitionen sowie Anweisungen beinhalten, diese können zur Verwendung mittels `import` in das vorgesehene Skript eingebunden werden [10]. Neben den Standardbibliotheken, die in der Regel Lösungen für alltägliche Probleme bieten, gibt es ebenfalls entsprechende Drittanbietermodule, welche von der Community entwickelt und auf dem Python Package Index (PyPI) [11] gelistet werden. Die folgenden Module wurden für diese Arbeit verwendet:

hashlib

Das Modul `hashlib` [12], aus der Standardbibliothek, bietet eine Reihe verschiedener Hashalgorithmen zur Verwendung an. Es umfasst die Algorithmen der SHA-Familie sowie

den RSA-MD5-Algorithmus. In dieser Arbeit wurde das Modul vor allem zum Zweck der Integritätsprüfung, also zur Sicherstellung der Nichtveränderung von Daten und Dateien, eingesetzt.

pipenv

Das Drittanbietermodul `pipenv` [13] erstellt und verwaltet virtuelle Entwicklungsumgebungen, die auf verschiedenen Systemen unterstützt werden. Dabei kann es mithilfe eines `Pipfile` [14] andere Module installieren oder deinstallieren.

pytest

Das Drittanbietermodul `pytest` [15] ist ein weitverbreitetes Testingframework, das unter anderem mit dem `assert` Schlüsselwort einen erwartenden Wert mit einem generierten Wert auf Gleichheit überprüfen kann. Dieses Modul wurde vor allem zum Nachweis der Integritätsprüfung verwendet.

pytsk3

Das Drittanbietermodul `pytsk3` [16] fungiert als Schnittstelle zu The Sleuth Kit (TSK) und ermöglicht den Zugriff auf die TSK-API [17]. TSK [18] selbst ist ein Kommandozeilen-Tool, das es ermöglicht, Disk-Images, vorwiegend auf Volumen- und Dateisystemdaten zu forensischen Zwecken, zu untersuchen. Die forensischen Analysen in dieser Arbeit wurden großteils mit diesem Modul durchgeführt.

filetype

Das Drittanbietermodul `filetype` [19] ermöglicht die Bestimmung des Dateityps und MIME-Typs mittels Überprüfung der Magic Numbers [20] einer Datei oder Puffers. Die Schnittstelle unterstützt eine Vielzahl unterschiedlicher Dateitypen, wie beispielsweise JPG, PDF oder auch Worddateien.

Weitere Module aus der Standardbib

Zur weiteren Realisierung der Skripte wurden neben `hashlib`, unter anderem `argsparse` [21] zur Übergabe von Argumenten von der Kommandozeile aus, `os` [22] zur Gewährleistung portabler Nutzung betriebssystemabhängige Funktionalitäten zu nutzen, sowie `csv` [23] zur Generierung und Befüllung von CSV-Dateien und `datetime` [24] zur Formatierung von Datum und Uhrzeit.

3.2.3 Automatisierungstool: Jenkins

Parallel zu Python gilt Jenkins ebenso als eines der beliebtesten Automatisierungstools im Kontext von Continuous Integration und Delivery (CI/CD), wie JetBrains in ihrer jährlichen Umfrage [25] aus dem Jahr 2023 erneut bestätigt. Jenkins bietet eine Vielzahl von Drittanbieteranbindungen als Plugins [26] an, welche es ermöglichen, eine Jenkins-Instanz je nach Anwendungsfall spezifisch zu konfigurieren. Jenkins stammt aus der Javawelt und ermöglicht zusätzlich die Verwendung von Groovy [27], die abgeleitete Skriptsprache von Java.

Jenkins Configuration as Code

Das Plugin Jenkins Configuration as Code (JCasC) [28] ermöglicht die Konfiguration codebasiert mit YAML-Syntax textuell aufzubereiten. Dies ermöglicht eine vollständige automatisierte Bereitstellung einer Jenkins-Instanz, wie so sonst nur mittels manueller Ausführung auf der grafischen Benutzeroberfläche durchführbar wäre. So können beispielsweise initial Jobs vordefiniert oder auch Plugins vorinstalliert werden. Diese Schritte lassen sich dann zur weiteren Optimierung in ein Shellskript überführen, um so eine vollständige Automatisierung herbeizuführen.

Jenkins Pipeline Syntax

Die Automatisierungsausführung, oft als Pipeline [29] ausgedrückt, wird für gewöhnlich in sogenannte `Jenkinsfiles` ausgelagert. Abbildung 3.3 stellt dabei den Aufbau dar, dabei beinhaltet eine `Pipeline` entsprechende `Stages` und diese wiederum `Steps`, welche die sequentiellen Ausführungsschritte beinhaltet.

```
pipeline {
    agent any
    stages {
        stage('Stage 1') {
            steps {
                //
            }
        }
        stage('Stage 2') {
            steps {
                //
            }
        }
          ...
    }
}
```

Abbildung 3.3: Deklarative Jenkins Pipeline Syntax

Kapitel 4

Implementierung

Dieses Kapitel stellt die Umsetzung basierend auf Kapitel 3 vor.

4.1 Konfiguration

Die Komponenten auf Abbildung 4.1 entstammen dem Tech Stack auf Abbildung 3.2 und sind entsprechend farblich gekennzeichnet.

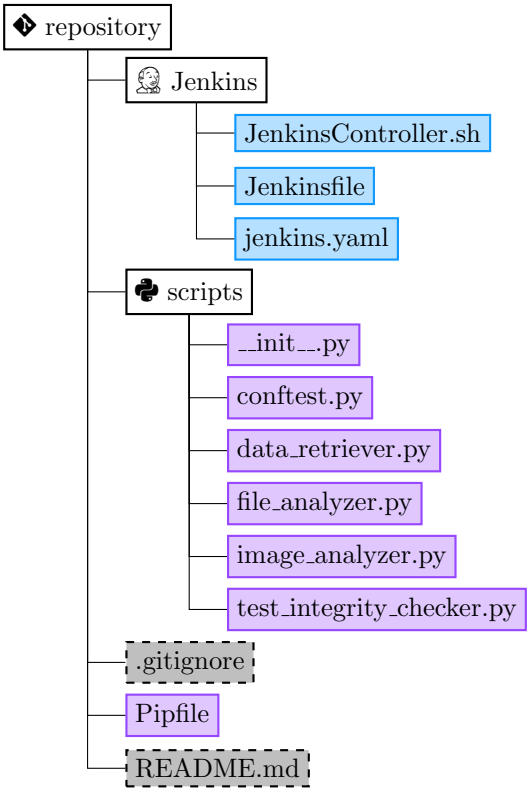

Abbildung 4.1: Repository

4.1.1 Python Konfiguration

Um eine automatisierte Entwicklungsumgebung des pipenv Modules zu gewährleisten, wird standardmäßig ein Pipfile verwendet. Hier werden alle Drittanbietermodule mit ihrer entsprechenden Version festgehalten und können mittels `pipenv install` initialisiert werden. Abbildung 4.2 zeigt das Pipfile für dieses Projekt.

```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
pytsk3 = "==20231007"
filetype ="==1.2.0"

[dev-packages]
pytest = "~=8.2.0"
```

Abbildung 4.2: Pipfile

4.1.2 Jenkins Konfiguration

Die Jenkins Konfiguration gestaltet sich in zwei Schritten:

```
jobs:
- script: >
    pipelineJob('forensics-automation') {
        parameters {
            stringParam('imagePath', '', 'path to image')
            stringParam('initialHash', '', 'images hash value')
        }
        definition {
          cpsScm {
            scm {
              git {
                remote {
                  url('./forensics-automation.git')
                }
                branches('*/main')
              }
            }
            scriptPath('Jenkins/Jenkinsfile')
          }
        }
    }
```

Abbildung 4.3: JCasC Yaml Datei

JenkinsController.sh

Das Shellskript ermöglicht die automatisierte Konfiguration der Jenkins-Instanz. Dabei werden Plugins unter Verwendung der `jenkins-cli` [30] installiert und JCasC aktiviert.

jenkins.yaml

Für dieses Projekt war es das Ziel, mittels JCasC und dem Job DSL Plugin [31] bereits den Job zur Durchführung der forensischen Analysen ausführbar bereitgestellt zu haben, sobald der Nutzer initial den lokalen Jenkins aufruft. Dies wird durch JCasC, mittels der Definition in Abbildung 4.3 dargestellt, ermöglicht. Des Weiteren wird der Build parametrisierbar gemacht, durch die Definition eines `parameter{}` Blocks [32]. So kann von außen dynamisch bei jeder Ausführung der Pfad zu dem zu untersuchenden Image, sowie dessen initialer Hashwert übergeben werden.

4.2 Skripte

Die forensichen Analysen unterteilen sich in die folgenden Phasen und wurde mittels Pythonskripten umgesetzt:

4.2.1 Integritätsprüfung

Die Integritätsprüfung wurde im Skript `test_integrity_checker.py` realsiert. Die Testdatei verwendet die Module pytest sowie hashlib und erhält als Parameter den Pfad zur Imagedatei sowie den zu erwartenden Hashwert als String. Damit die Parametrisierung gewährleistet werden kann, werden sogenannte Fixtures [33] aus dem pytest-Modul verwendet, die in der `conftest.py` definiert werden und implizit von allen Testdateien erkannt werden. Durch die Verwendung des hashlib Moduls berechnet die Funktion

```
def calculate_hash(test_object, hash_algorithm='md5', chunk_size=8192):
    hash = hashlib.new(hash_algorithm)
    with open(test_object, 'rb') as f:
        while True:
            data = f.read(chunk_size)
            if not data:
                break
            hash.update(data)
    return hash.hexdigest()


def test_compare_hash(get_test_object, get_initial_hash):
    assert calculate_hash(get_test_object) == get_initial_hash
```

Abbildung 4.4: Funktionen zur Hashwertgenerierung und anschließender Validierung

`calculate_hash`, dargestellt in Abbildung 4.4 den aktuellen Hashwert der Imagedatei. Weitere Argumente legen den Hashalgorithmus, nämlich MD5, sowie die chunk-size auf 8192 Bytes fest. Bei der chunk-size handelt es sich um ein Vielfaches von 128 Bytes. Dadurch kann die Berechnung speichereffizient erfolgen, da die Datei nie vollständig, sondern blockweise dem MD5 übergeben wird [34]. Schließlich wird der errechnete Wert der `test_compare_hash` übergeben und mit dem Parameter auf Gleichheit überprüft. Das

Skript gewährleistet durch diese Struktur eine effiziente und zuverlässige Integritätsprüfung von Imagedateien in einer automatisierten Testumgebung.

4.2.2 Image Analyse

Ziel dieser Analysephase ist es, relevante Informationen bezüglich des Images zu erhalten. Hierbei sollen die einzelnen Partitionen identifiziert und deren Dateien und Verzeichnisse gesammelt werden. Erhaltene Informationen werden schließlich in CSV-Dateien dokumentiert. Dies wurde mit dem Skript `image_analyzer.py` durchgeführt. Das Skript verwendet die Module os, pytsk3, argparse, datetime und csv. Als Parameter werden der Image-Pfad sowie der Output-Pfad der CSV-Dateien übergeben.

Volumen Analyse

Sobald das Image geöffnet ist können mittels `pytsk3.Volume_Info` Informationen zu den Partitionen gesammelt werden [35]. Dies erfolgt über das TSK_VS_PART_INFO Struct [36] und extrahiert wie in Abbildung 4.5 dargestellt, die relevanten Informationen der einzelnen Partitionen und speichert diese als Dictionary-Einträge in eine Liste und überführt der gesammelten Information mithilfe einer weiteren Funktion in eine CSV-Datei.

```
def get_partition_info(imagehandle):
    partition_info_list = []
    partitionTable = pytsk3.Volume_Info(imagehandle)
    for partition in partitionTable:
        try:
            partition_info = {
                'Partition': partition.addr,
                'Type': partition.desc.decode(),
                'Start': partition.start,
                'Size': partition.len,
                'Flag': partition.flags
            }
            partition_info_list.append(partition_info)
        except Exception as e:
            print(f"Unable to process partition: {e}")
    return partition_info_list
```

Abbildung 4.5: Funktion zur Extraktion der Partionensinformationen

Dateisystemanalyse

Sobald die Partitionen erkannt und deren Startsektoren identifiziert wurden, geht es in diesem Schritt darum, das zugehörige Dateisystem zu öffnen und darauf befindliche Dateien und Verzeichnisse auszulesen und zu dokumentieren. Dieser Schritt erfolgt iterativ, wobei der Startsektorwert jeder Partition multipliziert mit 512 Bytes als Offset entsprechend übergeben wird. Der Wert ist so gewählt, da ein Segment für gewöhnlich dieser Größe entspricht, wie auf Seite 31 in [1] dargestellt. Die Funktion `list_directories` durchläuft

rekursiv das übergebene Dateisystemobjekt und extrahiert mithilfe des TSK_FS_META Struct [37] die Metadaten der darin enthaltenen Dateien und Verzeichnisse. Die Metadaten umfassen Pfad, Inode, Name, Größe und Erstellungszeitpunkt, letzteres geeignet formatiert mithilfe des datetime Moduls. Beim Zugriff der einzelnen Partitionen ist davon auszugehen ist, dass es sich um unterschiedliche Dateisysteme handelt. Demzufolge wird das os Modul verwendet, welches mittels `os.path.join` eine betriebssystemunabhängige Verknüpfung von Dateipfaden ermöglicht, indem es automatisch den korrekten Pfadtrenner für das jeweilige Betriebssystem verwendet sowie unnötige Pfadtrenner vermeidet [38]. Nachdem die Informationen der jeweiligen Partitionen gesammelt wurden, wird als Nächste ebenfalls iterativ die entsprechenden CSV-Dateien generiert.

```
def list_directories(filesystem_object, directory_path="/"):
    file_metadata_list = []
    ...

    for entry in directory:
        entry_name = safe_decode(entry.info.name.name)
        if entry_name not in [".", ".."]:
            full_path = os.path.join(directory_path, entry_name)
            if entry.info.meta and entry.info.meta.type == pytsk3.TSK_FS_META_TYPE_DIR:
                try:
                    file_metadata_list.extend(
                        list_directories(filesystem_object, full_path))
                except OSError as e:
                    print(f"Unable to  access subdirectory {full_path}: {e}")
            else:
                try:
                    if entry.info.meta:
                        file_object = filesystem_object.open(full_path)
                        meta_info = file_object.info.meta
                        file_metadata = {
                            'Path': full_path,
                            'Inode': meta_info.addr,
                            'Name': entry_name,
                            'Size': meta_info.size,
                            'Creation Time': datetime
                                .datetime.fromtimestamp(meta_info.crtime).strftime(
                                '%Y-%m-%d %H:%M:%S')
                        }
                        file_metadata_list.append(file_metadata)
                except Exception as e:
                    print(f"Unable to access file metadata for {full_path}: {e}")
    return file_metadata_list
```

Abbildung 4.6: Funktion zur Extraktion der Verzeichnisseinträge

4.2.3 Wiederherstellung aktiver Daten

Dieser Schritt soll schließlich die erkannten Dateien auf den jeweiligen Partitionen wiederherstellen, um diese für weitere Schritte analysierbar zu machen. Dies erfolgt im Skript `data_retriever.py`, welches neben den bereits bekannten Argumenten, um den Partitionsstartsektorwert erweitert wird, um so aktiv zu entscheiden, welche Partition wiederhergestellt werden soll. Dabei verwendet das Skript die bereits beschriebene Funktion

`list_directories`, um die entsprechenden Metadaten als Liste der Funktion `save_files` zu übergeben. Diese Funktion iteriert dabei über jedes Element in dieser Liste, extrahiert den entsprechenden Dateipfad und versucht die Datei zu öffnen und ihre Daten zu lesen. Die Funktion `read_random` [39] entstammt dem pytsk3 Modul und ermöglicht es, eine Datei zu öffnen. Um zu gewährleisten, den kompletten Inhalt zu erreichen, wird der Lesebereich von Byte 0 bis zum Größenwert der Datei aus den Metadaten angegeben. Dessen Inhalt wird, spezifiziert durch 'wb', schließlich im Binärmodus in neue Dateien geschrieben, was zur Folge hat, dass Daten genauso geschrieben werden, wie sie gelesen werden [40].

```
def save_files(filesystem_object, file_metadata, output_directory):
    for metadata in file_metadata:
        file_path = metadata['Path']
        try:
            file_object = filesystem_object.open(file_path)
            file_data = file_object.read_random(0, file_object.info.meta.size)
            output_path = os.path.join(output_directory, file_path.strip("/"))
            os.makedirs(os.path.dirname(output_path), exist_ok=True)
            with open(output_path, 'wb') as f:
                f.write(file_data)
            print(f"File saved to: {output_path}")
        except Exception as e:
            print(f"Unable to open or read file {file_path}: {e}")
```

Abbildung 4.7: Funktion zur Wiederherstellung von festgestellten Dateien

4.2.4 Dateianalysen

Dateianalysen werden im Skript `file_analyzer.py` realisiert. Das Skript verwendet als weiteres neu hinzugekommenes Modul das filetype Modul. Nachdem die wiederhergestellten Dateien im entsprechenden Verzeichnis abgelegt wurden, können diese nun weiter analysiert werden. Beispielsweise empfiehlt es sich, Duplikate ausfindig zu machen, um so beispielsweise im Nachgang die manuellen Analysen effizienter zu gestalten. Dies wird in der Funktion `find_duplicates` gewährleistet. Dabei wird für jede Datei der Hashwert ermittelt und mit jeder anderen Datei verglichen. Entsprechende Treffer werden schließlich zur Dokumentation in csv-Dateien abgespeichert. Ebenfalls bietet es sich an, im Sinne der Correlation/Zusammenhang-Richtlinie, beschrieben in Abschnitt 3.1, gefälschte Dateien zu identifizieren oder generell Anomalien ausfindig zu machen. So identifiziert die `identify_file_types.py` mithilfe des filetypes Moduls die MIME-Typen der Dateien und ordnet sie den entsprechenden Dateien zu. Dies kann Aufschluss geben, ob es sich tatsächlich um den Dateityp handelt, wie ihre Dateiendung vorgibt. Eine weitere Funktion ist die `identify_large_files.py`, welche große Dateien mit einem fest definierten Schwellenwert vergleicht und alle Dateien, die diesen Wert überschreiten, entsprechend zu dokumentieren. Das Skript kann beliebig um ähnliche Funktionen erweitert werden.

4.3 Jenkins

Jenkins fungiert in dieser Arbeit als Bindeglied und führt dabei die beschriebenen Schritte in Abschnitt 4.2 sequentiell aus und speichert die gesammelten Informationen im Workspace der Jenkins Instanz ab. Die Wiederherstellung gelöschter Daten erfolgt über `foremost`. Der Prozess wurde auf der Kommandozeile ausgeführt und ebenfalls in Jenkins integriert. Das Jenkinsfile besteht aus neun Stages und beginnt damit die Python virtuelle Entwicklungsumgebung durch das `Pipfile` zu initialisieren, umso die entsprechenden Pythonskripte, wie in Abbildung 4.8 dargestellt, auf der Kommandozeile in den entsprechenden Stages auszuführen. Um den Kontrollfluss während der Ausführung zu

```
pipenv run python scripts/image_analyzer.py \
--image-path ${imagePath} --output-csv ${OUTPUT_DIR}/fileSystemAnalysis/

pipenv run pytest ./scripts/test_integrity_checker.py \
--test-object ${imagePath} --initial-hash ${initialHash}

pipenv run python scripts/data_retriever.py \
--image-path ${imagePath} --partitions-start ${value} \
--output-directory ${OUTPUT_DIR}/restoredData/activeData/partition${key}

pipenv run python scripts/file_analyzer.py \
--folder-path "${OUTPUT_DIR}/restoredData/" \
--subfolder-path "/activeData/partition2/
```

Abbildung 4.8: Pythonkomanndos zur Ausführung der Skripte auf der Kommandozeile

gewährleisten, werden um alle Befehle zum Ausführen der Skripte `catchError{}` Blöcke [41] gesetzt. Einzige Ausnahme stellen Integritätsprüfungen, falls eine dieser Stages fehschlägt, soll die Ausführung sofort unterbrochen werden. Durch Setzen dieser Blöcke, gewährleistet die Pipeline ihren Fortlauf, selbst wenn beispielsweise nicht alle Dateien einer Partition wiederhergestellt werden können. Dies kann unter anderem valide Gründe haben, wie beispielsweise ein beschädigtes Dateisystem. Der Nutzer kann darauf hingewiesen, indem man den Zustand der Stage beispielsweise auf UNSTABLE [41] setzt. In Jenkins besteht die Möglichkeit, eine `post` Sektion [42] unter verschiedenen Bedingungen zu definieren. Diese Sektion wird je nach Bedingung nach der Pipeline selbst aufgerufen. In diesem Jenkinsfile wurde die `always` Bedingung verwendet, was zur Folge, dass die Sektion immer ausgeführt wird, unabhängig vom Pipelineergebnis [42]. In dieser Sektion wird die `archiveArtifact` Methode [43] verwendet, welche das gewünschte Verzeichnis, hierbei das Verzeichnis, worin die gesammelten Informationen abgelegt werden, komprimiert und zum Download zur Verfügung stellt.

Kapitel 5

Ergebnisse

Dieses Kapitel stellt die Ergebnisse, welche sich basierend auf der Systematik aus Kapitel 3 und der Umsetzung aus Kapitel 4 ergeben haben.

5.1 Gesammelte Informationen

Abbildung 5.1 zeigt die Struktur der gesammelten Informationen und wie sie abgespeichert werden. Die vorliegenden Ergebnisse resultierten auf Anwendung auf das Image `forensicstick.dd`. Auf eine vollständige Darstellung in den folgenden Abschnitten wurde aufgrund der Länge der Ergebnisdateien verzichtet.

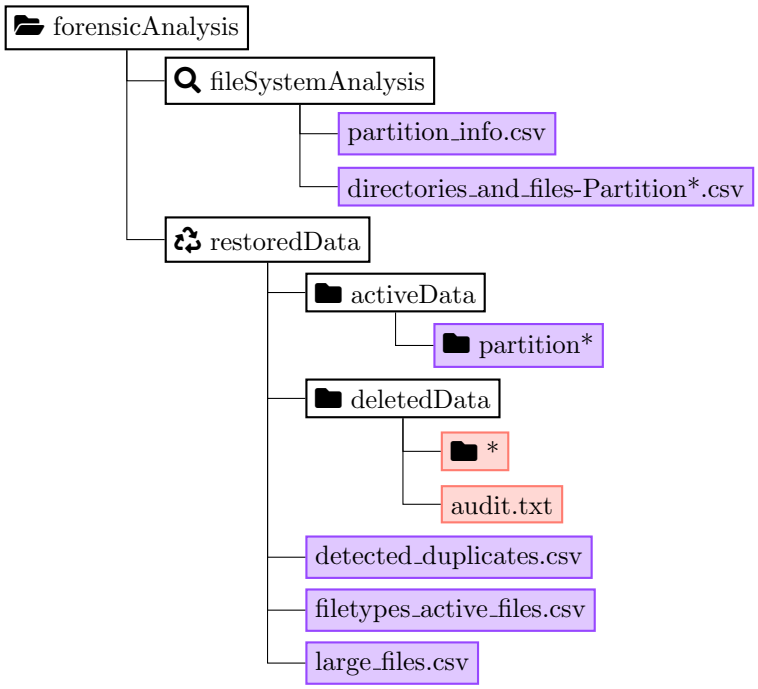

Abbildung 5.1: Darstellung der gesammelten Ergebnisse

5.1.1 Dateisystemanalyse

Die Tabelle 5.1 ergibt sich durch die Ausführung des Pythonskripts `image_analzyzer.py`. Dabei repräsentiert jede Zeile eine Partition und liefert weitere Informationen, wie die zugehörige Partitionsnummer, den Typ, den Startsektor, die Größe, sowie die entsprechende Kennzeichnung, repräsentiert durch ein Flag. Die Flags entstammen dabei dem enum

Partition	Type	Start	Size	Flag
0	Primary Table (#0)	0	1	4
1	Unallocated	0	32	2
2	NTFS / exFAT (0x07)	32	625632	1
3	Win95 FAT32 (0x0c)	625664	391168	1
4	Linux (0x83)	1016832	293376	1
5	Linux (0x83)	1310208	705024	1

Tabelle 5.1: Partitionsinformationen

TSK_VS_PART_FLAG_ENUM [44] welches die folgenden Werte, dargestellt in Tabelle 5.2, repräsentieren. Nachdem jede Partition identifiziert werden konnte, wurde mithilfe

Enumerator	Description
.ALLOC	Sectors are allocated to a volume in the volume system.
.UNALLOC	Sectors are not allocated to a volume.
.META	Sectors contain volume system metadata and could also be ALLOC or UNALLOC.
.ALL	Show all sectors in the walk.

Tabelle 5.2: Flagwerte des enums (entnommen aus [44])

der Funktion `list_directories` identifizierte Dateipfade extrahiert und als CSV-Dateien abgespeichert, welche als Auszug in Tabelle 5.3 dargestellt ist.

Path	Inode	Name	Size	Creation Time
/$AttrDef	4	$AttrDef	2560	2008-06-05 21:20:17
/$BadClus	8	$BadClus	0	2008-06-05 21:20:17
/$Bitmap	6	$Bitmap	78208	2008-06-05 21:20:17
/$Boot	7	$Boot	8192	2008-06-05 21:20:17
...	...	...	...	...

Tabelle 5.3: Auszug der Verzeichnisse und Dateien aus Partition 2

5.1.2 Datenwiederherstellung

Aktive Daten

Festgestellte Dateien aus Tabelle 5.3 wurden mittels der Funktion `save_files` aus dem Skript `data_retriever.py` wiederhergestellt und entsprechend der Abbildung 5.1 gespeichert.

Gelöschte Daten

Gelöschte Dateien wurden mithilfe von Foremost wiederhergestellt. Abbildung 5.2 zeigt das Protokoll, welches standardmäßig von Foremost nach Datenwiederherstellung generiert wird. Die gelöschten Daten wurden entsprechend der Abbildung 5.1 gespeichert.

```
File: /Users/mario/Documents/Github/cfv/testObject/forensicstick.dd
Start: Sun May 26 13:17:22 2024
Length: Unknown

Num       Name (bs=512)          Size       File Offset     Comment

0:       00000491.jpg            3 KB           251774
1:       00000753.jpg           66 KB           385682
...
2508:    02007146.pdf           21 KB       1027658752
Finish: Sun May 26 13:17:37 2024

2509 FILES EXTRACTED

jpg:= 545
gif:= 402
bmp:= 1
htm:= 489
ole:= 2
zip:= 50
rar:= 3
exe:= 586
png:= 219
pdf:= 212
-----------------------------------------------------------------

Foremost finished at Sun May 26 13:17:37 2024
```

Abbildung 5.2: Auszug der generierten foremost audit.txt

5.1.3 Dateianalyse

Wiederhergestellte Dateien wurden mithife des Skripts `file_analyzer.py` analysiert und die Ergebnisse in den folgenden Tabellen sowie im Anhang dargestellt. Tabelle 5.4 protokolliert mittels der Funktion `identify_file_types` die festgestellten MIME-Typen und weist diese den vorliegenden Dateien zu. Tabelle 5.5 protokolliert Dateien gemäß der Funktion `identify_large_files`, die größer 10 MB entsprechen. Tabelle 5.6 protokolliert

Duplikate, welche gemäß der Funktion `find_duplicates` identifizert werden konnten. Dabei liefert die dritte Spalte ,,File Paths" den absoluten Dateipfad zu den Dateien, welche denselben Hashwert aufweisen.

MIME Type	**File Paths**
image/x-icon	/partition2/$UpCase
application/pdf	/partition2/VDATA/artobrigheim.pdf
image/jpeg	/partition2/VDATA/Last/problemlsgsim.jpg
application/gzip	/partition2/LiveSystemRH/linux-suspended-md5s.gz
application/x-bzip2	/partition2/LiveSystemRH/linux-suspended.tar.bz2
...	...

Tabelle 5.4: Auszug der festgestellten MIME Typen und Dateipfade aus Partition 2

File Path	**File Size**
/partition2/LiveSystemRH/linux-suspended.tar.bz2	106892127
/partition2/ForensicChallenge2008/dfrws2008-challenge.zip	94421088

Tabelle 5.5: Festgestellte große Dateien aus Partition 2

Hash	**Duplicate Count**	**File Paths**
15546b79697d344697f643750eee4034	2	...
d154c01ec99ee6fdc4c37bcb2936e9de	2	...
777030fae8ae0916e9dff2dae4c56751	2	...
a38b6b9ec3334ce11a83acf3ca449691	4	...
a3641de3d6181d4c28f8b28a5df7a71e	3	...
...	...	...

Tabelle 5.6: Auszug der Duplikate aus Partition 2

Kapitel 6

Fazit

Unter Betrachtung der entsprechenden Ergebnisse aus Kapitel 5 und Appendix A kann das realisierte Konzept auf der Basis der in Kapitel 3 festgelegten Werkzeuge und Ansätze als *proof of concept* angesehen werden.

Sicherlich besteht Raum für Optimierungen, insbesondere kann die Dateianalysephase noch weiter ausgebaut werden. Python bietet hierbei weitere Module an, welche für tiefere forensische Analysen verwendet werden können. So lässt sich beispielsweise mithilfe des `stegano` Drittanbietermoduls [45] problemlos Steganografieanalysen in den Prozess integrieren. Ebenfalls können Malwareanalyse mithilfe von Drittanbietermodulen, wie `yara` [46], welches die Yara Regeln verwendet, um Malware auf Computersystemen zu identifizierten, [47], unterstützt werden.

Folglich ist es möglich, je nach Bedarf und Anwendungsfall das Konzept zu erweitern. Diese Erkenntnis untermauert, warum Python als Programmiersprache für diese Szenarien eine gute Wahl ist und sich entsprechender Beliebtheit erfreut.

Zusammenfassend hat sich während der Entwicklung des Konzepts herauskristallisiert, wie zeitintensiv die Analyse von Disk-Images sein kann. Ein Werkzeug mit einer kurzen Buildzeit von etwa 41 Sekunden trägt dabei erheblich zur Effizienz der Analysearbeit bei.

Für künftige Arbeiten wäre es sicherlich interessant zu untersuchen, welches Potenzial die Automatisierung in anderen Bereichen der digitalen Forensik, wie der Netzwerkforensik, Mobilforensik oder Cloudforensik hat.

Anhang A

Zweites Testobjekt

Im Folgenden sind Auszüge der Ergebnisse einer Analyse dargestellt, die durch die Anwendung des Konzepts auf ein weiteres zufällig ausgewähltes Disk-Image generiert wurden. Dies unterstreicht die generische Implementierung, die unabhängig von der Struktur eines Disk Images enstanden ist. Das Image entstammt der Webseite: `https://dftt.sourceforge.net/` und entspricht dem ersten Gelisteten.

ext-part-test-2.dd

Tabelle A.1: Partitionsinformationen

Partition	Type	Start	Size	Flag
0	Primary Table (#0)	0	1	4
1	Unallocated	0	63	2
2	DOS FAT16 (0x04)	63	52353	1
3	DOS FAT16 (0x04)	52416	52416	1
4	DOS FAT16 (0x04)	104832	52416	1
5	DOS Extended (0x05)	157248	155232	4
6	Extended Table (#1)	157248	1	4
7	Unallocated	157248	63	2
8	DOS FAT16 (0x04)	157311	52353	1
9	Unallocated	209664	63	2
10	DOS FAT16 (0x04)	209727	52353	1
11	DOS Extended (0x05)	262080	50400	4
12	Extended Table (#2)	262080	1	4
13	Unallocated	262080	63	2
14	DOS FAT16 (0x06)	262143	50337	1

Tabelle A.2: Dateisysteminformationen der Partition 2

Path	Inode	Name	Size	Creation Time
/primary-1.txt	4	primary-1.txt	0	2003-07-24 15:00:18
/$MBR	831123	$MBR	512	1970-01-01 00:00:00
/$FAT1	831124	$FAT1	103936	1970-01-01 00:00:00
/$FAT2	831125	$FAT2	103936	1970-01-01 00:00:00
/$OrphanFiles	831126	$OrphanFiles	0	1970-01-01 00:00:00

Tabelle A.3: Duplikate Dateien

Hash	Duplicate Count	File Paths
e46c7fd80c9b4a6023ebfbca20ca8dc8	4	...
0c7ab568a8b1c41c282634f6c6daad7f	4	...

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] B. Carrier, *File System Forensic Analysis*. Addison-Wesley Professional, 2005.

[2] Cambridge Dictionary, "automation," 2023.
URL: `https://dictionary.cambridge.org/dictionary/english/automation` [Zuletzt aufgerufen am 25. Mai 2024].

[3] D. Hayes and M. Kyobe, "The adoption of automation in cyber forensics," in *Conference on Information Communications Technology and Society, ICTAS 2020, Durban, South Africa, March 11-12, 2020*, pp. 1–6, IEEE, 2020.

[4] Stack Overflow, "Developer Survey 2023 – Developer Experience: Processes, tools, and programs within an organization," 2023.
URL: `https://survey.stackoverflow.co/2023/#section-developer-experience-developer-experience-processes-tools-and-programs-within-an-organization` [Zuletzt aufgerufen am 25. Mai 2024].

[5] Stack Overflow, "Developer Survey 2022 – Developer Experience: Processes, tools, and programs within an organization," 2022.
URL: `https://survey.stackoverflow.co/2022/#section-developer-experience-developer-experience-processes-tools-and-programs-within-an-organization` [Zuletzt aufgerufen am 25. Mai 2024].

[6] Foremost, "Introdcution," 2024.
URL: `https://foremost.sourceforge.net/` [Zuletzt aufgerufen am 25. Mai 2024].

[7] Python, "Python," 2024.
URL: `https://www.python.org/` [Zuletzt aufgerufen am 25. Mai 2024].

[8] Stack Overflow, "Developer Survey 2023 – Programming, scripting, and markup languages," 2023.
URL: `https://survey.stackoverflow.co/2023/#section-most-popular-technologies-programming-scripting-and-markup-languages` [Zuletzt aufgerufen am 25. Mai 2024].

[9] C. Koch, K. Müller, and E. Sultanow, "Which programming languages do hackers use? A survey at the german chaos computer club," *CoRR*, vol. abs/2203.12466, 2022.

[10] Python, "Modules," 2024.
URL: `https://docs.python.org/3/tutorial/modules.html#modules` [Zuletzt aufgerufen am 25. Mai 2024].

[11] PyPI, "PyPI," 2024.
URL: `https://pypi.org/` [Zuletzt aufgerufen am 25. Mai 2024].

[12] Python, "hashlib — Secure hashes and message digests," 2024.
URL: `https://docs.python.org/3/library/hashlib.html#module-hashlib` [Zuletzt aufgerufen am 25. Mai 2024].

[13] PyPI, "pipenv," 2024.
URL: `https://pypi.org/project/pipenv/` [Zuletzt aufgerufen am 25. Mai 2024].

[14] Pipenv: Python Dev Workflow for Humans, "Pipfile & Pipfile.lock," 2024.
URL: `https://pipenv.pypa.io/en/latest/pipfile.html#pipfile-pipfile-lock` [Zuletzt aufgerufen am 25. Mai 2024].

[15] PyPI, "pytest," 2024.
URL: `https://pypi.org/project/pytest/` [Zuletzt aufgerufen am 25. Mai 2024].

[16] PyPI, "pytsk3," 2024.
URL: `https://pypi.org/project/pytsk3/` [Zuletzt aufgerufen am 25. Mai 2024].

[17] Brian Carrier, "The Sleuth Kit (TSK) Library User's Guide and API Reference," 2015.
URL: `https://www.sleuthkit.org/sleuthkit/docs/api-docs/4.11.1/` [Zuletzt aufgerufen am 25. Mai 2024].

[18] Brian Carrier, "The Sleuth Kit: Overview," 2023.
URL: `https://www.sleuthkit.org/sleuthkit/` [Zuletzt aufgerufen am 25. Mai 2024].

[19] PyPI, "filetype," 2024.
URL: `https://pypi.org/project/filetype/` [Zuletzt aufgerufen am 25. Mai 2024].

[20] Wikipedia contributors, "Magic number (programming) — Wikipedia, The Free Encyclopedia," 2024. `https://en.wikipedia.org/wiki/Magic_number_(programming)#Magic_numbers_in_files` [Zuletzt aufgerufen am 25. Mai 2024].

[21] Python, "argparse — Parser for command-line options, arguments and sub-commands," 2024. https://docs.python.org/3/library/argparse.html [Zuletzt aufgerufen am 25. Mai 2024].

[22] Python, "os — Miscellaneous operating system interfaces," 2024. https://docs.python.org/3/library/os.html [Zuletzt aufgerufen am 25. Mai 2024].

[23] Python, "csv — CSV File Reading and Writing," 2024. https://docs.python.org/3/library/csv.html [Zuletzt aufgerufen am 25. Mai 2024].

[24] Python, "datetime — Basic date and time types," 2024. https://docs.python.org/3/library/datetime.html [Zuletzt aufgerufen am 25. Mai 2024].

[25] JetBrains, "JetBrains: Developer Ecosystem Survey 2023 — Team Tools," 2023. https://www.jetbrains.com/lp/devecosystem-2023/team-tools/#ci_tools [Zuletzt aufgerufen am 25. Mai 2024].

[26] Jenkins, "Jenkins Plugins," 2024. https://plugins.jenkins.io/ [Zuletzt aufgerufen am 25. Mai 2024].

[27] Groovy, "Groovy Programming Language," 2024. https://groovy-lang.org/ [Zuletzt aufgerufen am 25. Mai 2024].

[28] Jenkins Plugins, "Configuration as Code," 2024. https://plugins.jenkins.io/configuration-as-code/ [Zuletzt aufgerufen am 25. Mai 2024].

[29] Jenkins, "Pipeline," 2024. https://www.jenkins.io/doc/book/pipeline/#pipeline-1 [Zuletzt aufgerufen am 25. Mai 2024].

[30] Jenkins, "Using the Jenkins CLI," 2024.
URL: https://www.jenkins.io/doc/book/managing/plugins/#install-with-cli [Zuletzt aufgerufen am 25. Mai 2024].

[31] Jenkins Plugins, "Job DSL," 2024.
URL: https://plugins.jenkins.io/job-dsl/ [Zuletzt aufgerufen am 25. Mai 2024].

[32] GitHub Pages – Jenkins Plugin Job DSL, "pipelineJob: parameters: stringParam," 2024.
URL: https://jenkinsci.github.io/job-dsl-plugin/#path/pipelineJob-parameters-stringParam [Zuletzt aufgerufen am 25. Mai 2024].

[33] pytest, "conftest.py: sharing fixtures across multiple files," 2024.
URL: https://docs.pytest.org/en/8.2.x/reference/fixtures.html#conftes

t-py-sharing-fixtures-across-multiple-files [Zuletzt aufgerufen am 25. Mai 2024].

[34] Stackoverflow, "Get the MD5 hash of big files in Python," 2019.
URL: https://stackoverflow.com/a/1131238 [Zuletzt aufgerufen am 25. Mai 2024].

[35] Brian Carrier, "The Sleuth Kit (TSK) Library User's Guide and API Reference: Volume Systems," 2015.
URL: https://www.sleuthkit.org/sleuthkit/docs/api-docs/4.3/vspage.html [Zuletzt aufgerufen am 25. Mai 2024].

[36] Brian Carrier, "The Sleuth Kit (TSK) Library User's Guide and API Reference: TSK_VS_PART_INFO Struct Reference: Public Attributes," 2015.
URL: https://www.sleuthkit.org/sleuthkit/docs/api-docs/4.2/structTSK__VS__PART__INFO.html [Zuletzt aufgerufen am 25. Mai 2024].

[37] Brian Carrier, "The Sleuth Kit (TSK) Library User's Guide and API Reference: TSK_FS_META Struct Reference: Public Attributes," 2015.
URL: https://www.sleuthkit.org/sleuthkit/docs/api-docs/4.3/structTSK__FS__META.html [Zuletzt aufgerufen am 25. Mai 2024].

[38] Python, "os.path — Common pathname manipulations," 2024.
URL: https://docs.python.org/3/library/os.path.html#os.path.join [Zuletzt aufgerufen am 25. Mai 2024].

[39] GitHub – py4n6/pytsk, "pytsk/tzsk3.h," 2024.
URL: https://github.com/py4n6/pytsk/blob/main/tsk3.h#L141 [Zuletzt aufgerufen am 25. Mai 2024].

[40] Python, "Built-in Functions: open," 2024.
URL: https://docs.python.org/3/library/functions.html#open [Zuletzt aufgerufen am 25. Mai 2024].

[41] Jenkins, "Pipeline: Basic Steps: catchError," 2024.
URL: https://www.jenkins.io/doc/pipeline/steps/workflow-basic-steps/#catcherror-catch-error-and-set-build-result-to-failure [Zuletzt aufgerufen am 25. Mai 2024].

[42] Jenkins, "Pipeline Syntax: Declarative Pipeline: Sections: post," 2024.
URL: https://www.jenkins.io/doc/book/pipeline/syntax/#post [Zuletzt aufgerufen am 25. Mai 2024].

[43] Jenkins, "Jenkins Core: archiveArtifacts," 2024.
URL: `https://www.jenkins.io/doc/pipeline/steps/core/#archiveartifacts-archive-the-artifacts` [Zuletzt aufgerufen am 25. Mai 2024].

[44] Brian Carrier, "The Sleuth Kit (TSK) Library User's Guide and API Reference: tsk_vs.h File Reference: Enumeration Type Documentation," 2015.
URL: `https://www.sleuthkit.org/sleuthkit/docs/api-docs/4.3/tsk__vs_8h.html#a4b2397da0861c68e1c6f5101ce3dd8dc` [Zuletzt aufgerufen am 25. Mai 2024].

[45] PyPI, "stegano," 2024.
URL: `https://pypi.org/project/stegano/` [Zuletzt aufgerufen am 25. Mai 2024].

[46] PyPI, "yara," 2024.
URL: `https://pypi.org/project/yara/` [Zuletzt aufgerufen am 25. Mai 2024].

[47] GitHub – VirusTotal, "yara," 2024.
URL: `https://github.com/virustotal/yara` [Zuletzt aufgerufen am 25. Mai 2024].