Cloud Virtual Machine Forensics - An Anti-forensic Perspective

A thesis submitted to University of Hyderabad in partial fulfilment

for the degree of

Doctor of Philosophy

by

Sanda Pranitha

Reg. No. 19MCPC01

SCHOOL OF COMPUTER AND INFORMATION SCIENCES

UNIVERSITY OF HYDERABAD

HYDERABAD -500046

Telangana

India

March, 2023

CERTIFICATE

This is to certify that the thesis entitled **"Cloud Virtual Machine Forensics - An Anti-forensic Perspective"** submitted by **Sanda Pranitha** bearing **Reg. No. 19MCPC01** in partial fulfilment of the requirements for the award of **Doctor of Philosophy** in **Computer Science** is a bonafide work carried out by him under my supervision and guidance. This thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for award of any degree or diploma. Parts of this thesis have been published online in the following publications:

1. Sanda P, Pawar D, Radha V. **An insight into cloud forensic readiness by leading cloud service providers: a survey**. *Computing*. 2022 Apr 17:1-26. https://doi.org /10.1007/s00607-022-01077-2. [**Indexed: SCI, SCIE, SCOPUS, DBLP, UGC-CARE List(India)**]. This publication is reported as part of Chapter 2.

2. Sanda P, Pawar D, Radha V. **Blockchain-based tamper-proof and transparent investigation model for cloud VMs**. *The Journal of Supercomputing*. 2022 May 25:1-29. https://doi.org/10.1007/s11227-022-04567-4. [**Indexed: SCIE, SCOPUS, UGC-CARE List(India)**]. This publication is reported as part of Chapter 5.

and

has made presentations in the following conferences.

1. Sanda P, Pawar D, Radha V. **VM Anti-forensics: Detecting File Wiping Using File System Journals**. In *ICCET 2022*, International Conference on Computing in Engineering & Technology 2022 (pp. 497-508). Springer, Singapore. https://doi.org/10.1007/978-981-19-2719-5. [**Indexed: SCOPUS, EI, DBLP**]. This publication is reported as part of Chapter 3.

2. Sanda P, Pawar D, Radha V. **WiDeS: Wiping Detection using System-calls - An Anti-forensic Resistant Approach**. In 2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (*TrustCom*) (pp. 1695-1703) **(Core ranking conference)**. https://doi.ieeecomputersociety.org/10.1109/TrustCom60117.2023.00231 [**Indexed: EI**]. This publication is reported as part of Chapter 3.

Further, the student has passed the following courses towards fulfilment of coursework requirement for Ph.D.

	Course Code	**Name**	**Credits**	**Pass/Fail**
1	CS402	Algorithms	4	Pass
2	CS800	Research Methods in Computer Science	4	Pass
3	CS803	Data structures and Programming Lab	2	Pass
4	CS858	Ethical Hacking and Computer Forensics	3	Pass

Dr. Digambar Pawar
Supervisor
SCIS, University of Hyderabad
Hyderabad-500 046, India

Dr. Radha Vedala
Co-Supervisor
IDRBT
Hyderabad-500 057, India

Prof. Atul Negi
Dean of School
SCIS, University of Hyderabad
Hyderabad-500 046, India

DECLARATION

I, **Pranitha Sanda**, hereby declare that this thesis entitled **"Cloud Virtual Machine Forensics - An Anti-forensic Perspective"** submitted by me under the supervision of **Dr. Digambar Pawar** and **Dr. Radha Vedala**, is a bonafide research work and is free from plagiarism. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma. I hereby agree that my thesis can be deposited in Shodganga/INFLIBNET.

A report on plagiarism statistics from the University Librarian is enclosed.

Date:

Signature of the Student

(Pranitha Sanda)

Reg. No.: 19MCPC01

//Countersigned//

Signature of the Supervisors

(Dr. Digambar Pawar)

(Dr. Radha Vedala)

Dedicated To My Family & Teachers

Acknowledgements

I am deeply indebted to **Dr. Digambar Pawar** and **Dr. Radha Vedala**, my thesis supervisors for their meticulous guidance, wisdom, and support during the course of my Ph.D. at University of Hyderabad in collaboration with Institute for Development and Research in Banking Technology. Their consistent encouragement and positive reinforcement have made it a gratifying experience. I am especially grateful for their emphasis on time management. Their guidance in meeting deadlines and prioritizing tasks has not only improved the quality of my research but has also equipped me with essential skills for life beyond academia. Moreover, I am indebted to them for their insights into balancing personal and professional life. Their encouragement to maintain a healthy equilibrium has been transformative, teaching me to nurture relationships, pursue passions, and prioritize well-being alongside scholarly pursuits. I extend my heartfelt thanks to **Dr. Digambar Pawar** and **Dr. Radha Vedala** for their support, wisdom, and mentorship. Their influence has left an indelible mark on my academic journey and beyond.

I thank my doctoral review committee members, Assoc.Prof. Y.V. Subba Rao and Asst.Prof. P. Syam Kumar for their encouragement, insightful comments, and questions, which strengthened my knowledge.

It is my privilege to thank **Prof. Atul Negi**, Dean, SCIS, University of Hyderabad for his considerate support and encouragement throughout the tenure of my research work and for extending the facilities to pursue my research. It is my privilege to thank **Dr. Deepak Kumar** Director, IDRBT for his considerate support and encouragement throughout the tenure of my research work and for extending the facilities to pursue my research.

I extend my heartfelt thanks to each member of IDRBT for fostering a seamless and rewarding atmosphere that made my tenure at the institute truly enjoyable. I am grateful to every individual in my lab for contributing to a wonderful experience during my time here.

I especially thank my family members and friends for their support and inspiration during the Ph.D work. I would like to express my deepest gratitude to my mother, **Rajitha Sanda**, for her unwavering support both personally and professionally. She has been my greatest support system throughout this journey. Without her, this thesis would not have been possible. She selflessly took care of my children, allowing me the time and peace of mind to focus on my research and writing. Her love, encouragement, and belief in me have been the foundation upon which all my achievements stand. Thank you, Amma, for everything. I am also profoundly grateful to my father, **Prabhakar Sanda**, for his tireless efforts and sacrifices. His dedication in ensuring I had the necessary support to pursue my studies has been invaluable. His sacrifices and belief in me have been a constant source of motivation. Thank you, Pappa, for your boundless support and for always being there for me. Lastly, I would like to thank my husband, **Prashanth Durki**, and acknowledge my children, **Yashika** and **Gitanshi**, who are my strength and motivation. Their love and patience have inspired me to persevere, and their smiles have been a constant reminder of why I embarked on this journey. Thank you for your understanding and for being my greatest source of joy and purpose.

At the end, I am grateful to **University of Hyderabad** and **Institute for Development and Research in Banking Technology** (IDRBT), for making it a memorable experience.

Abstract

Cloud has become indispensable due to its meteoric increase in utilization. An increase in the utilization of cloud resources has also fueled an increase in cyber incidents in the cloud. This alarming increase in cloud incidents emphasizes the need for the readiness of CSP for cloud forensics and anti-forensics. Handling anti-forensics in the cloud VMs is imperative, as it is mostly overlooked. Anti-forensic approaches used in traditional systems are equally applicable in cloud VMs, and artifact wiping is one such approach. The world has witnessed artifact wiping as a Wiper attack in recent warfare between Israel and Hamas, Ukraine, and Russia. Wiper attacks are considered destructive malware as they cause permanent damage to data. Thus, detecting, mitigating, and restoring the effect of such anti-forensic techniques used in cloud incidents becomes crucial to ensure the completeness of the evidence collected for investigation.

Despite the challenges, identifying and countering anti-forensic techniques in a cloud VM also offers advantages. Monitoring cloud VM activity is generally easier compared to detecting user activity on personal computers due to limited visibility and privacy concerns in personal and non-regulated environments. This is because the cloud typically offers centralized administration with monitoring tools, providing administrators with comprehensive visibility into VM activities, resource usage, and performance metrics. These integrated tools can be used to streamline the monitoring process and enable proactive detection and mitigation of anti-forensic techniques.

Our research objective is to address the anti-forensic practice of artifact wiping in the cloud VMs. It is proposed to provide a solution to detect wiping and withstand it by restoring the wiped contents from cloud VMs by exploiting the underlying file system data recovery mechanism. Further, we propose an investigation model to preserve

the cloud VMs and investigation findings. For this, we have considered the existing cloud forensic frameworks by the leading CSPs, i.e., Amazon, Azure, and Google Cloud, as well as other challenges associated with cloud forensics. We address these challenges by incorporating blockchain technology in our model to ensure immutability, transparency, and integrity of the evidence and investigation proceedings across the stakeholders involved in the investigation.

Contents

Acknowledgments **vi**

Abstract **viii**

List of Figures **xiv**

List of Tables **xvi**

1 Introduction **1**
1.1 Digital Forensics 1
1.2 Anti-forensics 2
1.2.1 Artifact Wiping 3
1.2.2 Gaps in Existing Literature for Wiping 4
1.3 Cloud Forensics 4
1.3.1 Gaps in Existing Literature for Cloud Forensics 5
1.4 Research Objectives 6
1.5 Scope and Problem Definition 6
1.6 Contribution of the Thesis 7
1.6.1 Detection of Wiping Activity 7
1.6.2 Recovery of Wiped Files 8
1.6.3 Preservation of Evidential Artifacts 9
1.7 Organization of the Thesis 9

2 Background and Literature Survey **11**
2.1 Anti-forensics 11
2.1.1 Data Hiding 11
2.1.2 Trail Obfuscation 12
2.1.3 Artifact Wiping 12

2.1.3.1 Types of Artifact Wiping 13
2.1.3.2 Related Work for Wiping Detection 13
2.1.3.3 Limitations 14
2.2 Cloud Forensics 15
2.2.1 Cloud Forensics by the Leading CSPs 17
2.2.1.1 Forensic Workflow of Leading CSPs 18
2.2.1.2 Evaluating Cloud Forensic Challenges Across Proposed Solutions by Leading CSPs 22
2.2.2 Limitations of Cloud Forensics 23
2.3 Summary 24

3 Detecting File Wiping **25**
3.1 Challenges in Existing Literature 25
3.2 Contributions 26
3.3 Prelimnaries 26
3.3.1 Data Sanitization 26
3.3.2 Ext File System Journaling 27
3.3.3 System Calls 28
3.4 Detecting Wiping Using File System Journals and Data Blocks . . 28
3.4.1 Proposed Model WiDeJ 28
3.4.1.1 Ext Journal Analysis 29
3.4.1.2 Using Shanon's Entropy 32
3.4.2 Results and Discussion 36
3.5 Detecting Wiping using System-calls 41
3.5.1 Role of System-calls in Behavior Analysis 42
3.5.2 Proposed Model WiDeS 42
3.5.2.1 Profiling Process Behaviour 43
3.5.2.2 Filtering Driven by Write() System-call 52
3.5.2.3 Analysis of Buffer Data Entropy 53
3.5.3 WiDeS Workflow 55
3.5.4 Results and Discussion 57
3.6 Summary 61

4 Recovery of Wiped Files **63**
4.1 Challenges in Recovering Wiped Files 64
4.2 Contributions 66
4.3 Prelimnaries 66
4.3.1 BTRFS Chunks 66
4.3.2 BTRFS Trees 67
4.3.3 BTRFS Data Structures 67
4.4 Recovery Using Journals 71
4.5 Proposed Model ReWinD 72
4.5.1 ReWinD Using btrfs-progs 72
4.5.2 ReWinD by Logging PA of Files 78
4.6 Results and Discussion 90
4.7 Usecase: Recovery of a file encrypted by Gonnacry ransomware . 93
4.8 Summary 95

5 Investigation Model to Preserve Cloud VMs and Investigation Proceedings on Blockchain **97**
5.1 Challenges in Existing System 98
5.2 Contributions 99
5.3 Prelimnaries 100
5.3.1 Blockchain 100
5.3.2 Hyperledger Fabric (HLF) 100
5.4 Application of Blockchain in Digital Forensics 102
5.4.1 Log Integrity 102
5.4.2 Metadata Integrity 103
5.4.3 Chain of Custody 103
5.5 Proposed Model Investigation-Chain 105
5.5.1 Blockchain Participants 107
5.5.2 Investigation-Chain Workflow 113
5.6 Proof of Concept 115
5.6.1 Case Study (Child Pornography) 115
5.7 Results and Discussion 118
5.7.1 Analysis of computational cost and communication overhead 118
5.7.2 Comparative analysis of the recent research with Investigation-Chain 126
5.8 Summary 129

6 Conclusion and Future Work **130**
6.1 Summary of Contributions 130
6.1.1 List of Contributions 131
6.2 Future Work 132

References **138**

A Cloud Forensic Workflows of Leading CSP **149**
A.1 AWS Workflow 149
A.2 Azure Workflow 151
A.3 GCP Workflow 153

List of Figures

1.1 Phases in Digital Forensics 2
1.2 Objectives and Contribution of the Thesis 7

2.1 AWS Forensics Workflow 19
2.2 Azure Forensics Workflow 20
2.3 GCP Forensics Workflow 21
2.4 Thesis reseach direction in the context of anti-forensics in comparison to the existing systems 24

3.1 Journal after mounting the file system 29
3.2 Journal after copying a file 30
3.3 File's inode data captured in journal block 30
3.4 Block content of a file at specific block pointer (i.e 9216) 31
3.5 Journal after wiping a file 35
3.6 Block content after wiping a file with zero 35
3.7 Content at data block when wiped with random characters 36
3.8 Scope of recovering wiped file data blocks using file system journals by varying wiping tools and file size. 37
3.9 Difference between regular file and file wiped with 0s 38
3.10 Difference between regular file and file wiped with random characters 39
3.11 File layout (reproduced from [1]) 40
3.12 Subset of SoS for benign process 'ls' 45
3.13 Unique patterns of benign process 'ls' 45
3.14 Subset of SoS for wiping process 'Shred' 46
3.15 Unique patterns of wiping process 'Shred' 46
3.16 WiDeS Workflow 56
3.17 Descision tree for WiDeS 58
3.18 Benign and wiping processes entropies 59

4.1 BTRFS key and BTRFS item in leaf node 68
4.2 BTRFS header of chunk tree 69
4.3 BTRFS items captured for a chunk tree leaf node 70
4.4 Traversing Ext3 journal entries to recover previous versions of the file 71
4.5 Superblock output using btrfs-progs 73
4.6 Root backup from superblock 74
4.7 FS tree dump using btrfs-progs 76
4.8 File data after wiping 77
4.9 File Content before wiping 78
4.10 Superblock layout for bootstrapping the file system 80
4.11 Root tree and chunk tree LA from superblock 82
4.12 sys_arr_chunk from superblock 82
4.13 Chunk tree leaf node layout 84
4.14 Root Tree Leaf Node Layout 87
4.15 Fs tree leaf node layout 89
4.16 Comparing the scope of file recovery between Ext and BTRFS under different scenarios as listed in Table 4.3 91
4.17 Scope of recovering wiped files from BTRFS by varying the DATA-CHUNK and file sizes 93
4.18 Superblock after encryption 94
4.19 FS Tree before and after encrypting the file using Gonnacry . . . 95
4.20 File content at physical address before and after encryption 95

5.1 Existing Investigation Procedure. 99
5.2 Architecture of Investigation-Chain 107
5.3 Operational workflow of Investigation-Chain 114
5.4 Hash value of snapshot computed using Autopsy tool 116
5.5 Cloud CSP instance image file findings using Autopsy tool 117
5.6 Time vs size of the snapshot 120
5.7 tps vs avg latency and throughput for 5-organizations-1-peer . . 122
5.8 tps vs avg latency and throughput for 4-organizations-1-peer . . . 123
5.9 tps vs avg latency and throughput for 3-organizations-1-peer . . . 124
5.10 Average latency comparison for three network models 125
5.11 Throughput comparison for three network models 125

List of Tables

2.1 Evaluating Cloud Forensic Challenges across proposed solutions by Leading CSPs 23

3.1 Entropy values for different file types 38
3.2 Comparison of WiDeJ and [2] 41
3.3 Benign processes' patterns and their frequencies 48
3.4 Wiping processes' patterns and their frequencies 49
3.5 NEs of wiping and benign processes 51
3.6 Processes classified as wiping (from module 1) and their patterns with maximum frequencies 53
3.7 List of benign processes, falsely classified as wiping processes with high-frequency patterns 60

4.1 BTRFS trees and their object IDs 67
4.2 BTRFS Tree items 69
4.3 Experimental setup for different scenarios to evaluate BTRFS . . 91

5.1 Digital Forensic Solutions based on Blockchain: Summary of strengths and weakness 105
5.2 Operational cost in milliseconds 121
5.3 Avg latency and throughput for a network with 5-organizations-1-peer 122
5.4 Avg latency and throughput for 4-organizations-1-peer 123
5.5 Avg latency and throughput for 3-organizations-1-peer 124
5.6 Average resource utilization for 5-organization-1-peer for all transaction send rate 126
5.7 Digital Forensic Solutions based on Blockchain: A Comparative Analysis of security elements 128

Chapter 1

Introduction

1.1 Digital Forensics

Digital Forensics is a process done as a part of investigation post the occurrence of an incident to find evidence over digital media, it was initially developed for data recovery [3]. Digital forensics involves examining digital media related to an incident. This examination aims to uncover traces of evidence left on digital devices. These traces result from actions taken during the incident. Digital forensics involves five crucial phases, i.e., identification, collection, analysis, preservation, and presentation of digital evidence (see Figure 1.1) in a way that is admissible in a court of law. Digital forensics is not just limited to cyber incidents but applies to other regular criminal offenses as the involvement of digital media, such as computers, laptops, mobiles, IoT (Internet of Things) devices, etc, is unavoidable. It would be more appropriate to say that digital media is becoming the core component of investigation in many cases. Based on the type of evidence being examined, we classify digital forensics as cloud forensics, network forensics, multimedia forensics, and mobile forensics. It is multidisciplinary as it involves technology, law, psychology, etc. According to the National Crime Records Bureau (NCRB) report, India has witnessed a 24% increase in cybercrime cases recorded in 2022 compared to cases in 2021 [4]. The global reports show an increase in the digital forensic market by 10% by 2032 [5]. Even during the pandemic Covid 19 the world has witnessed the sudden rise of cyber incidents and an increase in investment in digital forensic tools and expertise, thus raising the scope of the need for digital forensics.

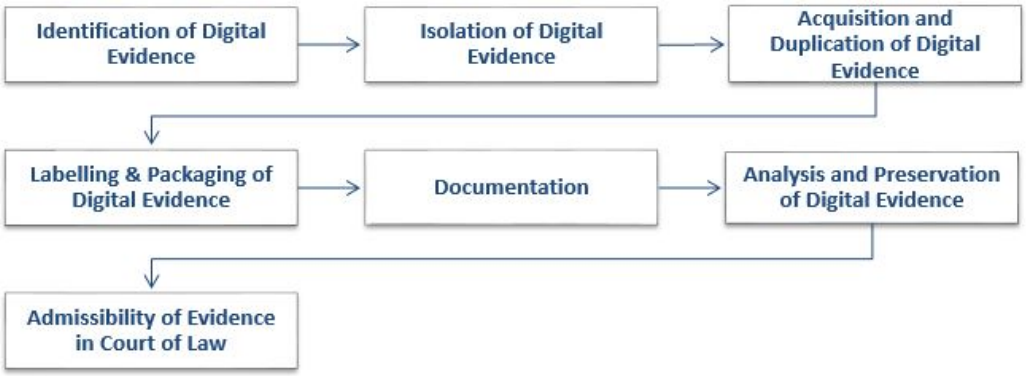


Figure 1.1: Phases in Digital Forensics

There are many challenges in digital forensics, including the rapid evolution of technology, global jurisdictions, lack of skilled forensic analysts, and many more; However, we would like to highlight a few of them that we believe are crucial: anti-forensics, cloud environment and virtualization, fragile nature of digital evidence.

1.2 Anti-forensics

Anti-forensics is a process used by malicious users to challenge the investigation procedure. It includes the tools and techniques to obstruct forensic analysis. Anti-forensics is an attempt to compromise the availability or usefulness of evidence to the forensic process [6]. Anti-forensics involves either destroying data or data misdirection, false/falsified data leading to wrong and inappropriate forensic analysis [7]. The adversary adopts anti-forensic techniques to interrupt the investigation proceedings, thereby leading to inappropriate or incomplete investigation findings. Encryption, artifact wiping, and data hiding are some of the well-known anti-forensic approaches. There has been limited research in anti-forensics, but it becomes crucial to detect anti-forensics early to ensure the completeness of investigation proceedings. Therefore, there is a pressing need to explore anti-forensics further.

We aim to explore artifact wiping, an anti-forensic technique that deletes data permanently by overwriting file content multiple times, making data recovery impossible. Also, it has not only been adapted as an anti-forensic approach but has also been used in wiper attacks. Wiper attacks are considered destructive as they target to cause permanent damage to data. Unlike other cyber attacks, the

motive of the wiper attack is not to get financial benefits but to affect the other involved party with irrecoverable damage. Wiper attacks are posing national level security threats and are exercised in cyber war fares [8], [9]. Thus, detecting such attacks and restoring the affected files becomes crucial to minimize the effect of the damage caused.

1.2.1 Artifact Wiping

In the context of anti-forensics, artifact wiping refers to the intentional secure deletion of activity traces from digital media. One cannot recover the file when it is securely deleted. Several tools, such as Eraser, Wipe, Sfill, SRM, etc., perform secure-delete tasks. Secure-delete is a suite of command-line utilities for securely deleting files on Linux systems. These utilities are designed to ensure that files are overwritten and irrecoverable, thus enhancing security and privacy. Secure deletion is practiced to ensure user data privacy regulations such as the General Data Protection Regulation (GDPR), Health Insurance Portability and Accountability Act (HIPPA), etc, but unfortunately, it is being practiced by adversaries to remove the traces of an incident. Based on the type of artifact wiped, it can be classified as 1) disk wiping and 2) file wiping. Our research focuses on detecting file wiping and restoring wiped files, with future work planned to address disk wiping.

File Wiping

A file is a collection of data stored as a single unit with a unique name and location within a file system. The data corresponding to the file is stored in blocks; they are consecutive sectors on the disk that store the file content. Each file occupies certain blocks based on its size. In Linux-based machines, an inode is the data structure that stores files' metadata (e.g., file size, file type, data block pointers, timestamps). We use an inode to locate the file on the disk. Upon deleting a file, the file's corresponding block pointer references from the inode are removed; because of this, we shall not be able to access the file directly. However, the file's content is still available in the data blocks. Thus, it is possible to recover the deleted files because in file deletion, the metadata corresponding to the file is deleted, but the actual file content is still available on the disk. Unfortunately, as the file system ages, deleted files cannot be recovered as new data overwrites the unallocated space containing their content. However, if a file is only partially overwritten, there is still a chance to recover part of it. Therefore, through the use of suitable forensic tools, we can potentially retrieve deleted files either in full

or partially. On the other hand, wiping is a secure deletion method. It involves not only removing metadata information from the inode but also overwriting the content of the corresponding file with random or specific characters (such as 0s or 1s), or repeating patterns of characters multiple times, resulting in the complete loss of data associated with the file.

1.2.2 Gaps in Existing Literature for Wiping

- The current literature has placed relatively less emphasis on anti-forensics when compared to digital forensics. However, it becomes crucial to detect anti-forensics even before starting the investigation to ensure the investigation is complete.
- The existing literature speaks only about detecting artifact wiping but does not show the procedure to restore them. Whereas, restoring wiped files could be very crucial as files serves as the basic artifact for any case.
- Current literature detects wiping using the signatures of the tools used to perform anti-forensics. But as the signature of the tools evolve these tools go undetected by the forensic tools.

1.3 Cloud Forensics

Cloud forensics is extending the application of digital forensics, which oversees the crime committed over the cloud and investigates it [10]. It emphasizes on collection, analysis, preservation, and presentation of cloud evidential artifacts as admissible in a court of law.

In this thesis, we would like to address the challenges corresponding to file wiping in cloud Virtual Machines (VMs); this can also be extended to cloud storage solutions, personal systems, or systems in the corporate environment. We focus on the cloud environment as we see it as an opportunity for the benefit of law enforcement agencies and Cloud Service Providers (CSPs) to curb anti-forensics in cloud VMs. Cloud environments, being centrally monitored and administered, present an opportunity to deploy our scripts for detection and recovery across targeted VMs efficiently. These scripts can operate in the background, continuously monitoring artifact-wiping. However, while we acknowledge the advantages of

cloud technology in combating anti-forensics, it is imperative to recognize the associated challenges in cloud forensics. Digital forensics, when extended to cloud resources, considers the challenges in the cloud, such as virtualization, multi-tenancy, data privacy, geographical jurisdictions, etc., and is known as cloud forensics. In cloud forensics, one of the crucial and primary evidential artifacts are cloud VMs. Cloud VMs are pivotal in cloud forensics due to their role as primary data carriers, the availability of snapshots for preserving evidence, and the ability to leverage cloud's scalable and remote capabilities. Hence, in this thesis, we explore cloud VM forensic challenges and present appropriate solutions for preserving cloud VMs in a sound forensic manner.

1.3.1 Gaps in Existing Literature for Cloud Forensics

- Increased response time to identify the malicious behavior in the context of file wiping in VMs and mitigate the effect of an incident.
- Dependency on CSP for the evidential artifacts. Law Enforcement Agency (LEA) has to depend on CSP, unlike traditional cybercrime cases. Ownership of evidence is with CSP, who cannot be trusted completely.
- Recovery of deleted or overwritten data. Due to the multi-tenant nature of the cloud most of the data is either overwritten to support other customer requirements or deleted due to data retention policy by CSP. If any delay occurs in sending the notice to preserve the evidence, then the crucial data required for investigation may be lost. Also, the forensic tools available for the cloud have many limitations in the retrieval of deleted or overwritten data.
- Trust on CSP and data integrity. Evidence stored in the cloud is always vulnerable to insider threats; it can be done intentionally to protect a company's reputation or by a former company employee.
- Lack of Standard Operating Procedure (SOP). There is no single valid approved process for digital forensics, this may challenge LEA in their investigation proceedings. Moreover, each CSP may have its own forensic process, which may or may not be valid.

- Forensic data acquisition and preservation. Due to a lack of cloud forensic tools, trainers, and trained professionals, forensically acquiring data and preserving it without compromising the integrity of evidence is a challenge.

1.4 Research Objectives

The main objective of this thesis is to uncover file wiping in cloud VMs. In this direction, the three main objectives of the thesis include;

1. Detection of wiping activity in cloud VMs.
2. Recovery of wiped files in cloud VMs.
3. Preservation of the cloud VMs.

1.5 Scope and Problem Definition

Cloud has become indispensable due to its meteoric increase in utilization. Gartner's prediction shows a 70% increase in the utilization of cloud resources by 2027, which currently in 2023 is around 15% [11]. A report by Thales [12] shows that 75% of businesses save 40% of their sensitive data on the cloud; it also, states that cloud assets are the biggest targeted resources for cyberattacks as 39% of businesses suffer from data breaches on the cloud. This alarming increase in cloud cyber attacks emphasizes the need for the readiness of CSP for cloud forensics. At the same time, we should also be prepared to detect, mitigate, and restore the effect of any anti-forensic techniques used in cyber-cloud attacks. Anti-forensic approaches applicable in traditional systems are equally applicable in cloud environments, artifact wiping is one such approach. The world has witnessed artifact wiping as a Wiper attack in recent warfare between Israel and Hamas, Ukraine, and Russia, also Saudi Arabia energy sectors were targeted [8], [9], [13].

Our objective is to walk through the challenges of cloud forensics while simultaneously addressing the anti-forensics approaches like artifact wiping in the cloud VMs. Further, devised an investigation model to collect and preserve the cloud evidential artifacts such that it ensures transparency and integrity of the investigation proceedings across the stakeholders involved in the investigation.

1.6 Contribution of the Thesis

The objective of the stated research has been discussed in three different aspects of file wiping i.e., detection, recovery, and preservation. This section briefly elaborates on the contributions towards each objective. Figure 1.2 provides the big picture of the contributions aligned with the objectives of the thesis.

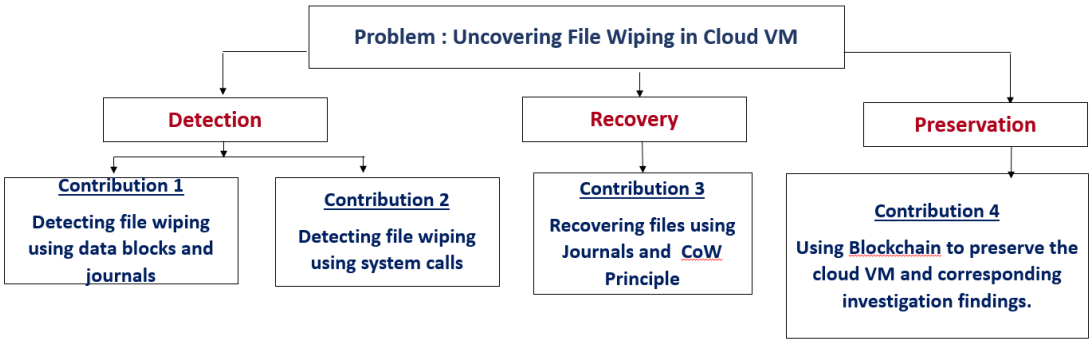


Figure 1.2: Objectives and Contribution of the Thesis

1.6.1 Detection of Wiping Activity

The first objective of the thesis focuses on detecting wiped files using the two approaches. The first approach **(Contribution 1)** includes wiping detection using file system journals, data blocks, and information theory metrics, and the second approach **(Contribution 2)** includes wiping detection using a Sequence of system calls and information theory metrics. In the first approach, we use file system journals to locate wiped file's data blocks on the disk. Further, we determine the block content byte-by-byte at each block allocated to the file. Likewise, we consider all blocks allocated to the file. We look at the frequency distribution of characters at these blocks and detect file wiping using Shannon's entropy.

In the second approach, we detect wiping using the sequence of system calls and information theory metrics and analyze the behavior of the benign and wiping process. Early study shows that system calls played a significant role in classifying normal process behavior from malicious process behavior [14], [15], [16]. We detect wiping by profiling the benign process behavior and the wiping process behavior. We monitor the system behavior at the process level by using short patterns from the sequence of system calls invoked by these processes. We observe how often a pattern is repeated in a process. Processes with a high

frequency of patterns indicate characteristic behavior associated with the wiping process's activity. As wiping involves overwriting the contents multiple times, the same patterns of system calls are invoked multiple times, thereby increasing the probability of patterns in the wiping process. Conversely, processes that exhibit infrequent patterns that reflect diverse or random patterns with less probability are often found in benign processes. Thus, we can classify the process based on the degree of randomness of patterns; here, we use Shanons' entropy to measure the randomness of patterns.

1.6.2 Recovery of Wiped Files

The second objective of the thesis focuses on restoring wiped files using file system recovery mechanisms. We recovered wiped files using two different file systems i.e., 1) Journal-based file system (Extended file system (Ext)) and 2) Copy-on-Write (CoW) based file system (B-Tree file system (BTRFS) **(Contribution 3)**. The existing literature has discussed detecting wiping using signature-based approaches but does not discuss restoring the wiped files. We restore the wiped files using both journaling in the Ext(3/4) file system and CoW in BTRFS.

Journals are used to track the changes committed on the disk even before they are executed on the disk to ensure file system consistency. In case of system failures, the journal is replayed to restore the file system. In the process of recovery, the committed transactions on the journal are executed on the disk, and the incomplete transactions are rolled back to the previous state. Thus, by traversing the journal back we can restore the previous version of the file. However, due to the cyclic queue data structure of the journal, the journal contents are overwritten by itself. Thus, if the journal contents are not recovered in time we may not be able to restore the wiped file.

On the other hand, in a CoW-based file system, when a file is updated or modified, instead of overwriting the existing file, the file's content is copied to a new location, and the changes are done at the new location [17]. Thus, the original file remains unchanged. This allows multiple file versions to be stored simultaneously, which can be useful for keeping track of changes made over time. If the operating system itself overwrites the file we cannot recover the older version of the file. However, this limitation is also applicable to other file systems as well.

1.6.3 Preservation of Evidential Artifacts

The third objective of the thesis focuses on using blockchain to preserve the evidential artifacts (like cloud VM) and investigation findings of the cloud incidents. Blockchain is a promising technology that will ensure data integrity, immutability, trust, and transparency among multiple stakeholders. To ensure the integrity of evidence in the cloud, most of the researchers in this domain have proposed applying blockchain on cloud forensic artifacts (i.e., cloud logs, chain of custody, metadata) [18], [19]. Thus, we propose an investigation model to preserve cloud VMs and their findings on blockchain **(Contribution 4)**.

Summary of Contributions

- An approach to detect file wiping on virtual machines using disk contents (i.e., virtual disk) and information theory metrics.
- A novel approach to detect file wiping on virtual machines using the sequence of system calls and information theory metrics.
- An approach to restore the wiped files using file system journals.
- A novel approach to restore the wiped files using CoW-based file system BTRFS.
- A usecase for extending our proposed model for recovery of encrypted files using Gonnacry application.
- A big picture of cloud forensic approaches adopted by the leading CSPs.
- An investigation model to preserve evidential artifacts cloud VM that is tamper-proof and transparent across the stakeholders involved in the investigation.

1.7 Organization of the Thesis

The contributions presented as part of the thesis entitled "Cloud Virtual Machine Forensics- An Anti-forensics Perspective" are structured into six different chapters. In this section, a brief overview of the chapters is illustrated. The

four contributions are presented in Chapter 3 (which includes two contributions), Chapter 4, and Chapter 5.

Chapter 1. Introduction: A brief introduction to the thesis is given in this chapter. It introduces digital forensics, anti-forensics, and cloud forensics. It also briefly identifies the research gap and establishes the motivations to achieve the objectives of the thesis. Finally, the chapter is concluded with the organization of the thesis followed by the list of contributions to the thesis.

Chapter 2. Related Work: In this chapter, we discuss current literature and the challenges in the context of the research objectives of this thesis.

Chapter 3. Detecting File Wiping: In this chapter, we propose a method to detect file wiping in cloud VMs using two approaches: 1. Using journals, data blocks, and information theory metrics, 2. Using system calls and information theory metrics. The first approach is to detect wiping post-occurrence of the incident, and the second approach is to detect wiping when the incident is active.

Chapter 4. Recovery of Wiped Files: In this chapter, we propose an approach to recover wiped files from cloud VMs using the file system's data recovery mechanisms. Data recovery mechanisms are used to restore the file system to a consistent state in case of power failure, system crashes, or hardware errors. Two major data recovery mechanisms used in file systems include 1) Journaling and 2) CoW. In this chapter, we present two different approaches for file recovery using these file system recovery procedures. Additionally, we present a usecase for recovery of unencrypted file versions following a ransomware attack launched by Gonnacry.

Chapter 5. Investigation Model to Preserve Cloud VMs and Investigation Proceedings on Blockchain: In this chapter, we detail the procedure of preserving the cloud VMs and the investigation findings using a blockchain. The proposed model also addresses additional challenges in cloud forensics, such as trust on CSP, transparency in the investigation, collusion across the involved stakeholders, and integrity of evidence.

Chapter 6. Conclusion and Future Scope : This chapter summarizes the contributions made to address the aforementioned objectives to achieve the overall problem statement with substantial evidence. Finally, this chapter concludes with future research directions.

Chapter 2

Background and Literature Survey

In Chapter 1, we discussed digital forensics, anti-forensics in the context of artifact wiping, and how the cloud can be considered as an opportunity for the proposed solutions to detect wiping, recover wiped content, and preserve the evidential artifacts. We also listed the gaps and challenges corresponding to anti-forensics and cloud forensics. In this chapter, we provide the background and related research in the context of anti-forensics and cloud forensics.

2.1 Anti-forensics

Anti-forensics is a process used by malicious users to challenge the investigation procedure. It includes the tools and techniques to obstruct forensic analysis. According to Harris [6], anti-forensics is an attempt to compromise the availability or usefulness of evidence to the forensic process. Anti-forensics involves either destroying data or data misdirection, false/falsified data leading to wrong and inappropriate forensic analysis [7]. Based on early studies [6], [20], [21], [22], [23], [24] we generalized anti-forensic techniques broadly as data hiding, trail obfuscation, and artifact wiping.

2.1.1 Data Hiding

Data hiding is an approach adapted intentionally to hide the data inorder to bypass the investigation findings. The authors in [6], [20], and [21] lists many

approaches for data hiding. Commonly used data-hiding approaches include encryption and steganography [21]. Encryption is altering the plain text into the encoded text to avoid unauthorized access. In digital forensics, encryption is used by adversaries to prevent the readability of evidential artifacts unless an appropriate decryption key is available. The authors of [25], [26], and [27] details the impact of encryption on digital forensics and the procedure to recover encrypted files. Steganography is the way to hide data in other files like documents, images, audio, and videos for privacy and confidentiality. Also, steganography is mostly known for illicit purposes by adversaries to hide evidential artifacts from direct access. The authors in papers [28], [29] show the application of steganography and its detection for forensic investigation.

2.1.2 Trail Obfuscation

Trail obfuscation is the practice of misleading the investigator intentionally by altering the evidential artifacts or destroying the evidential artifacts, thereby making investigation difficult or impossible. It predominantly includes metadata manipulations, IP/MAC address spoofing, proxy servers, log cleaners, compression bomb, DDoS attacks, etc [21], [30]. By altering the metadata, they mislead the investigator resulting in inappropriate timeline analysis and event reconstruction. IP/MAC address spoofing, proxy server, and P2P networks ensure anonymity and conceal the adversary's identity. Log cleaners remove the traces of the trails left behind upon execution of cyber incidents. Compression bombs and DDoS attacks exploit the computational resources, thereby causing the system to crash or freeze, hindering and delaying the investigation.

2.1.3 Artifact Wiping

Artifact wiping typically involves overwriting the content stored with random data multiple times, making it extremely difficult to recover. The act of wiping files itself may be evidence of criminal intent. Detection and analysis of artifact wiping may be used to establish the intent and motive of the cyber incident. From the existing anti-forensics techniques, wiping is the commonly adapted technique [21], [31], because it is supported by many commercial and freeware that is easy to install and use. The authors in paper [31] explain the integration of anti-forensics approaches in other attacks whose primary purpose is to delete evidence (e.g., Wiper attacks). A report by Fortiguard shows an increase in the wiper

attacks by 50% [32]. This motivates us to consider artifact wiping as our research objective. Based on the type of artifact being wiped, it can be further classified as disk wiping and file wiping.

2.1.3.1 Types of Artifact Wiping

Based on the type of artifact wiped, it can be classified as 1) disk wiping and 2) file wiping. In this research, we have investigated file wiping and restored wiped files.

Disk Wiping:
Disk wiping overwrites the disk space on the storage media multiple times, making data recovery impossible. Here, disk space can be a few blocks allocated to a file, a disk partition, or the entire disk. It is used to ensure data protection regulations for organizations' securely deleting content while disposing of old system hard disks. The authors in paper [2] used the block contents to detect wiping on disk. It uses statistical methods such as entropy and a statistical test suite developed by NIST to determine the randomness of block content and detect wiped data fragments on disk compared to standard data fragments.

File Wiping:
We discussed file wiping in Chapter 1; it involves overwriting the contents of the targetted file. Unlike disk wiping, in file wiping, the adversary targets only specific files that may contain sensitive data crucial for investigation. Adversary adapts file wiping to minimize detection risk. It challenges forensic investigators, leaving the entire system intact and making it difficult to identify which specific file was wiped. Unlike disk wiping, file wiping doesn't draw one's attention. Also, file wiping is time and resource-efficient compared to disk wiping. Thus, adversaries prefer file wiping over disk wiping. Hence, it becomes crucial to detect file wiping and mitigate its effect.

2.1.3.2 Related Work for Wiping Detection

The existing techniques to detect wiping are based on the traces left by the wiping tool following their execution. We classified all such approaches as signature-based wiping detection models. The authors in [33] detect wiping based on the entries found in various metadata structures of the windows file system like FAT32 (File Allocation Table), glsexfat (Extended File Allocation Table), NTFS (New Technology File System). For FAT32 and exFAT, the directory structure entries were used, and for the NTFS file system, $MFT, $LogFile, $UsnJrnl files were

analyzed to detect wiping for four different wiping tools. Horsman in [34], used Digital Tool Marks (DTM) left behind by the seven different wiping tools following their usage to detect file wiping. He determined the impact of eight wiping tools on NTFS and FAT32 file systems. Wiping detection was done based on the differences identified between regular and wiped file's metadata using the file system artifacts $Logfile and $MFT. Unlike the above-mentioned papers, Joo et al. [35] used window artifacts instead of file system artifacts. Total 13 window artifacts (like Prefetch, AmCache, Jumplist, ShimCache, etc) were analysed by varying 10 different wiping tools. Further, they consolidated their results in a database for investigators quick reference. Park et al. [36] used signatures of an anti-forensic tool (eraser) under the action, i.e., the traces left during installation, execution, and uninstallation. The systems monitoring tool collects system logs. These logs are used to collect the signatures of the tools. The signatures captured are then compared with known anti-forensic tool signatures. If the signatures match, anti-forensics is detected.

Most studies have focused on detecting wiping but not on recovering wiped content. Recovery of wiped files will be of significant importance in the forensic investigation as it can provide crucial evidence. We could find papers [37], [38] that restore the previous versions of the file. Swenson et al. [37] restores the previous version of the file using the ExT file system journal, but due to the cyclic queue data structure of the Journal, the journal content gets overwritten. Thus, recovery of the previous file version is only possible upon restoring the journal at the appropriate time. Peterson et al. [38] enabled Ext file systems with CoW capabilities for file versioning. They used snapshots for versioning at the file level. This motivates us to use a CoW-based file system to explore further for recovery of wiped files.

2.1.3.3 Limitations

We explored file wiping and summarized the limitations observed based on the above discussion specific to file wiping below,

- To continuously be updated about the emerging wiping tools signature. If any signature of the tool is unknown, it goes undetected.
- If the wiping tools erase its traces from the referenced artifacts leaving no trace of its execution then detecting wiping becomes challenging.

- Most of the literature has focused on the detection of file wiping, but the scope of recovering the wiped files is yet to be explored.
- Shortfall of literature for investigating the recent file systems in use for detecting and recovering wiped files.

2.2 Cloud Forensics

Simson Garfinkel, in his paper [3], discussed the golden age of digital forensics and emphasized the need to upgrade the existing forensic tools for the next generation. By using traditional forensic tools, data residing in the computer's digital media can be retrieved, but these tools are not showing promising results when applied on cloud [3]. Dykstra, in his paper [39], presents two hypothetical case studies of crime committed over the cloud and further details the challenges encountered due to the usage of existing standard forensic tools used for the investigation of these cases. As there is a substantial increase in demand for cloud computing, the need for the tools addressing the challenges encountered in the cloud is needed.

A survey by K. Ruan [40] on critical criteria and definitions for cloud forensics publishes the results of a survey conducted across 257 forensic experts and practitioners. This survey focuses on fundamental questionnaires regarding cloud forensics, such as its definition, challenges, usage, significance, opportunities, required criteria, etc. The majority of the respondents supported cloud forensic definition as - "Cloud forensics is a mixture of traditional computer forensics, small-scale digital device forensics, and network forensics" and "Cloud forensics is an application of digital forensics in cloud computing." Increased usage of cloud technology has opened gateways for hackers to commit crimes over the cloud with much ease[41], at the same time increasing the risks and challenges in the investigation procedure for LEAs and CSPs. Investigation procedure, the nature of evidence, and many other artifacts corresponding to crimes committed over the cloud vary from traditional crime involving digital media. The Investigating Officer (IO) will neither have direct access to the physical resources nor will be involved in evidence collection due to various privacy constraints in the cloud. The IO has to depend on the CSPs for evidence completely. There is a lack of standard operating procedures for conducting forensics in the cloud as every crime over the cloud differs by cloud architecture, type of service, and deployment model being used [10].

There are multiple challenges in cloud forensics as the cloud supports multi-tenancy, infinite storage, virtualization, remote accessibility, etc. Thus, CSPs play a major role in the identification, collection, and preservation of evidence by ensuring the integrity of the evidence. Earlier published survey papers by researchers in this domain have detailed the challenges in cloud forensics. Also, the National Institute of Standards and Technology (NIST) has published two reports i.e., [42] and [43]. These technical reports elaborate an exhaustive list of challenges corresponding to architecture, data collection, data analysis, anti-forensics, trust in incident first responders (CSP), role management, standard operating procedures, training, and legal challenges. Survey paper by Bharat Manral [44] details cloud forensic challenges. It illustrates the work accomplished in cloud forensics and the challenges by categorizing them as incident-driven, provider-driven, consumer-driven, and resource-driven cloud forensics. Ameer Pichan, in his paper [45], details a comparative analysis of technical challenges and solutions in cloud forensics. This paper lists the challenges for each phase of the digital forensic process, i.e., identification, preservation, collection, analysis, and presentation. It provides a comparative analysis of available solutions proposed by researchers, NIST and Amazon. A survey paper by Shams Zawoad [10] summarizes the challenges in cloud forensics based on cloud service models, i.e., Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). It lists the earlier proposed forensic solutions for IaaS, PaaS, SaaS, and corresponding challenges. Ben Martini, in his paper [41], discusses cloud forensic technical challenges and reviews the proposed models by researchers. This paper details the challenges and areas of improvement in public and private cloud storage applications. K.Ruan, in her paper [46], discusses cloud forensics and elaborates on challenges in three different dimensions, i.e., technical, organizational, and legal. Further, Expert review on cloud forensic readiness framework by organizations [47], this study is motivated by a particular gap in research on the technical, legal, and organizational factors that facilitate forensic readiness in organizations that utilize an (IaaS) model.

The authors of the paper highlight the need for existing forensic procedures to withstand anti-forensics in the cloud [48], [49]. Radha, et al. [48] proposed a taxonomy for anti-forensics in cloud. They proposed a framework for cloud forensic investigation that included a module to check for anti-forensics before reporting and presenting the evidence. However, the authors didn't disclose the details of the implementation. Prasad and Vrushali [49], discussed the demand for a framework that mitigates anti-forensics in cloud forensics. Further, the author

also emphasizes the need for blockchain-based solutions to ensure the integrity of the investigation process due to the increased application of anti-forensics.

We observed that most of the papers on cloud forensics are focused on the cloud forensic challenges and contributions by the research community to address those challenges. However, we aim to understand the current cloud forensic procedure in practice by the leading CSPs and examine if the current procedures can address any of the existing challenges, further identify the gaps in real-time, and try to address the gap accordingly.

2.2.1 Cloud Forensics by the Leading CSPs

In this section, we compare the forensic readiness of leading CSPs. Based on Gartner, Inc. magic quadrant on leading CSPs published in 2020 [50], Amazon Web Services (AWS), Azure, and Google Cloud Platforms (GCP) are the leading CSPs. These CSPs are compared against various parameters based on the provisioning of cloud forensic procedures. Comparison parameters are selected based on essential cloud characteristics for collection, analysis, and preservation of evidential data for investigation. The information is obtained from the official websites of the respective CSPs, conferences, and workshops. On analysis of cloud forensic procedure of AWS [51] [52], Azure [53] [54] and GCP [55] it is found that the providers are implementing cloud forensics by performing, Log, Disk, and Memory Forensics. However, we could not find a supporting document regarding memory forensics by Azure, but as stated in the incident response document [54] that memory forensics is implemented by Azure.

- Log Forensics- Log forensics is the analysis of cloud logs to mitigate and investigate malicious behavior of the attacker [56]. The captured cloud logs are preserved and presented such that it is admissible and used in court trials.

- Disk Forensics- Disk forensics in the cloud involves creating the snapshot of the suspected virtual instance along with its attached disk volumes. It also involves ensuring the integrity of the acquired snapshots and analyzing the disk contents by using forensic tools, and presenting the admissible evidence to the court of law.

- Memory Forensics- Memory forensics is capturing and analyzing volatile data of the affected cloud resource (i.e., VM's Random Access Memory

(RAM)). The memory captured (memory dump) is analyzed preserved and presented to the court of law such that it is admissible. It is also known as live forensics.

2.2.1.1 Forensic Workflow of Leading CSPs

We analyzed that forensics in the cloud environment are carried out on three main cloud artifacts by these leading CSPs; they are logs, disk, and volatile memory of virtual instances involved in the incident. Thus, we analyzed the forensic procedures for log, disk, and memory forensics by leading CSPs by referring publicly available official documentation. Forensic workflow is the sequence of tasks that processes the evidential data in every step of each CSP and presents a bird's eye view of end-to-end forensic workflow for each CSP i.e. AWS forensic workflow in Fig 2.1, Azure Forensic workflow in Fig 2.2 and GCP forensics Workflow in Figure 2.3 (see Appendix A for a detailed description of the workflows by each CSP).

AWS Forensics Workflow

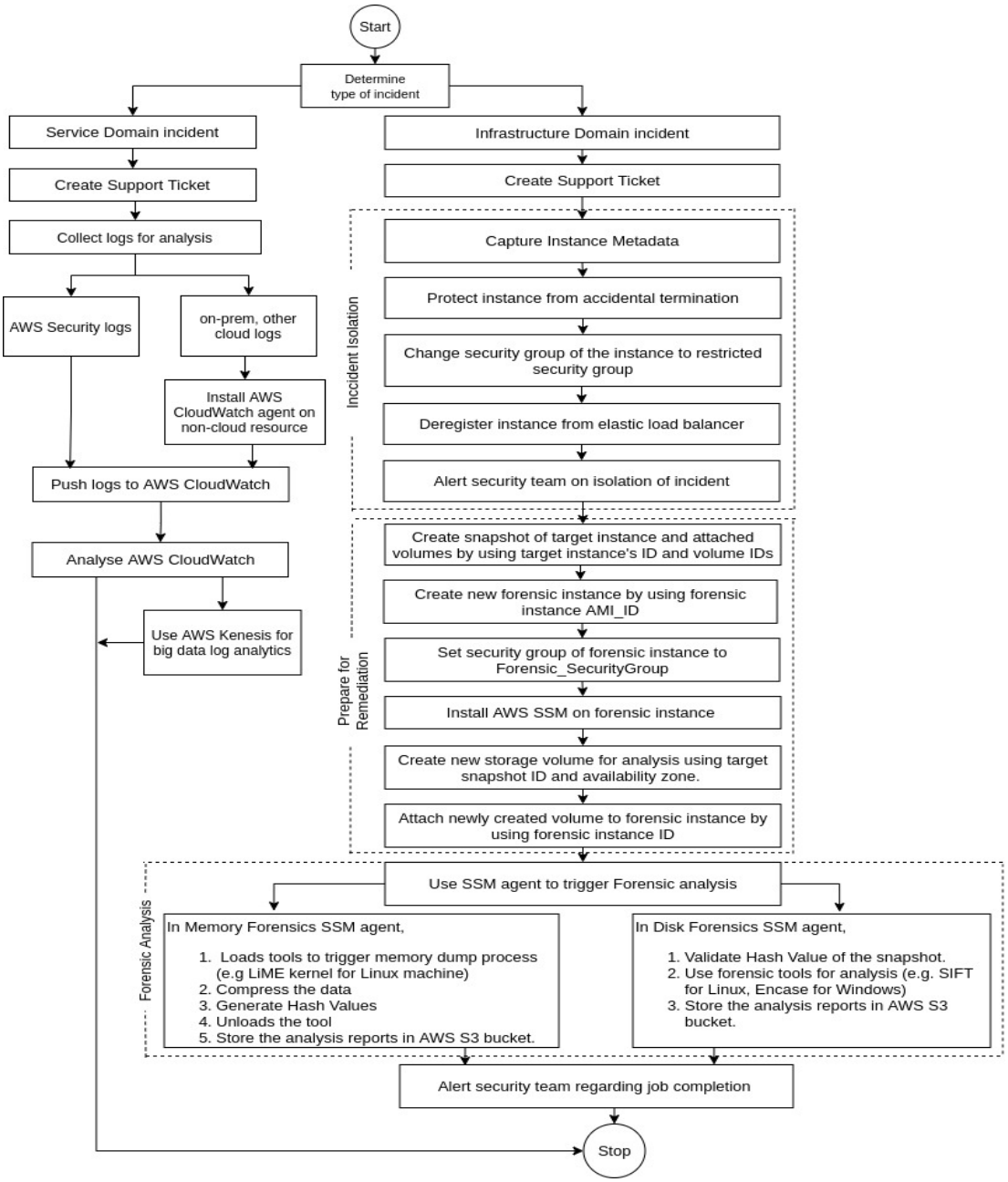


Figure 2.1: AWS Forensics Workflow

Azure Forensics Workflow

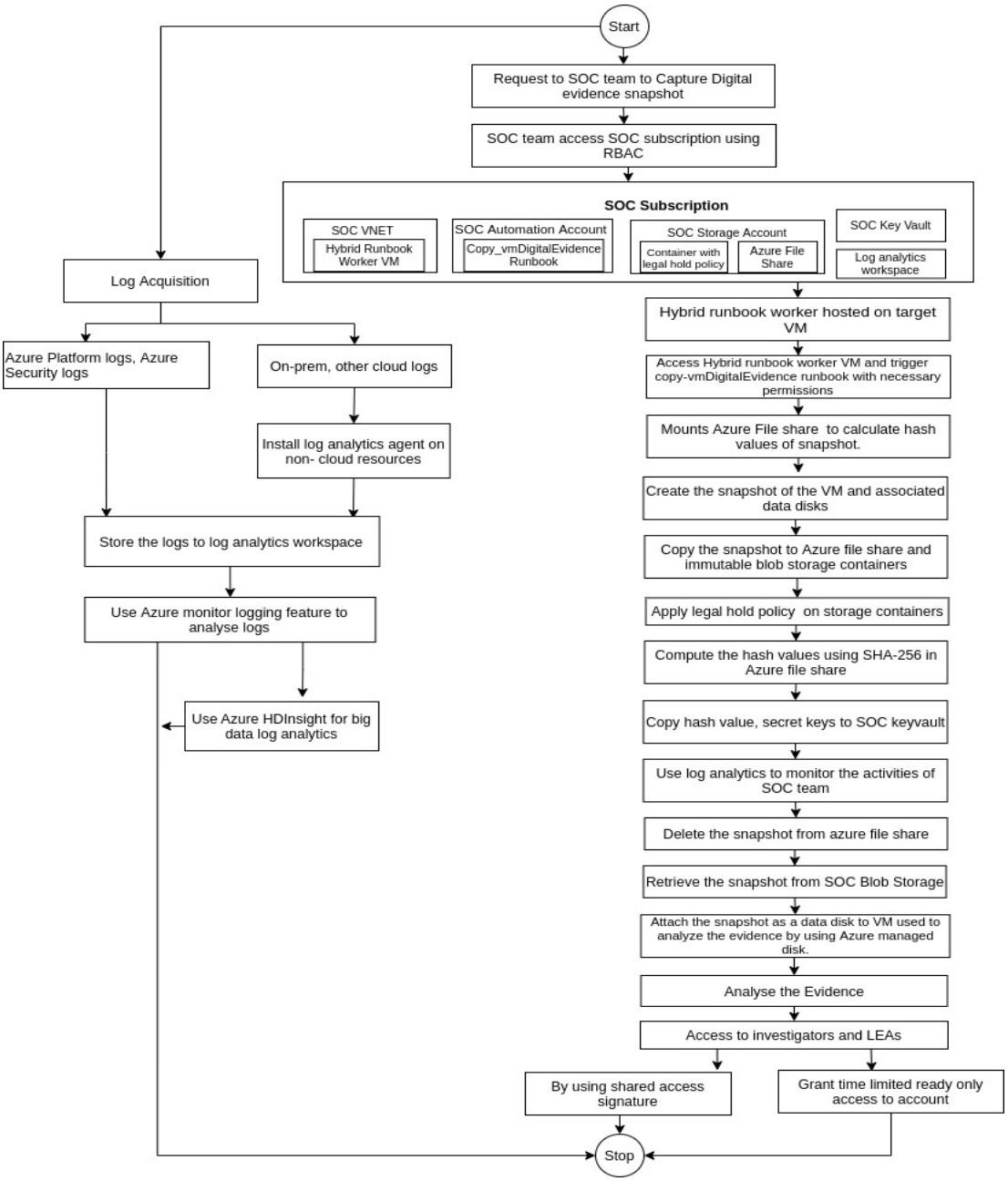


Figure 2.2: Azure Forensics Workflow

GCP Forensics Workflow

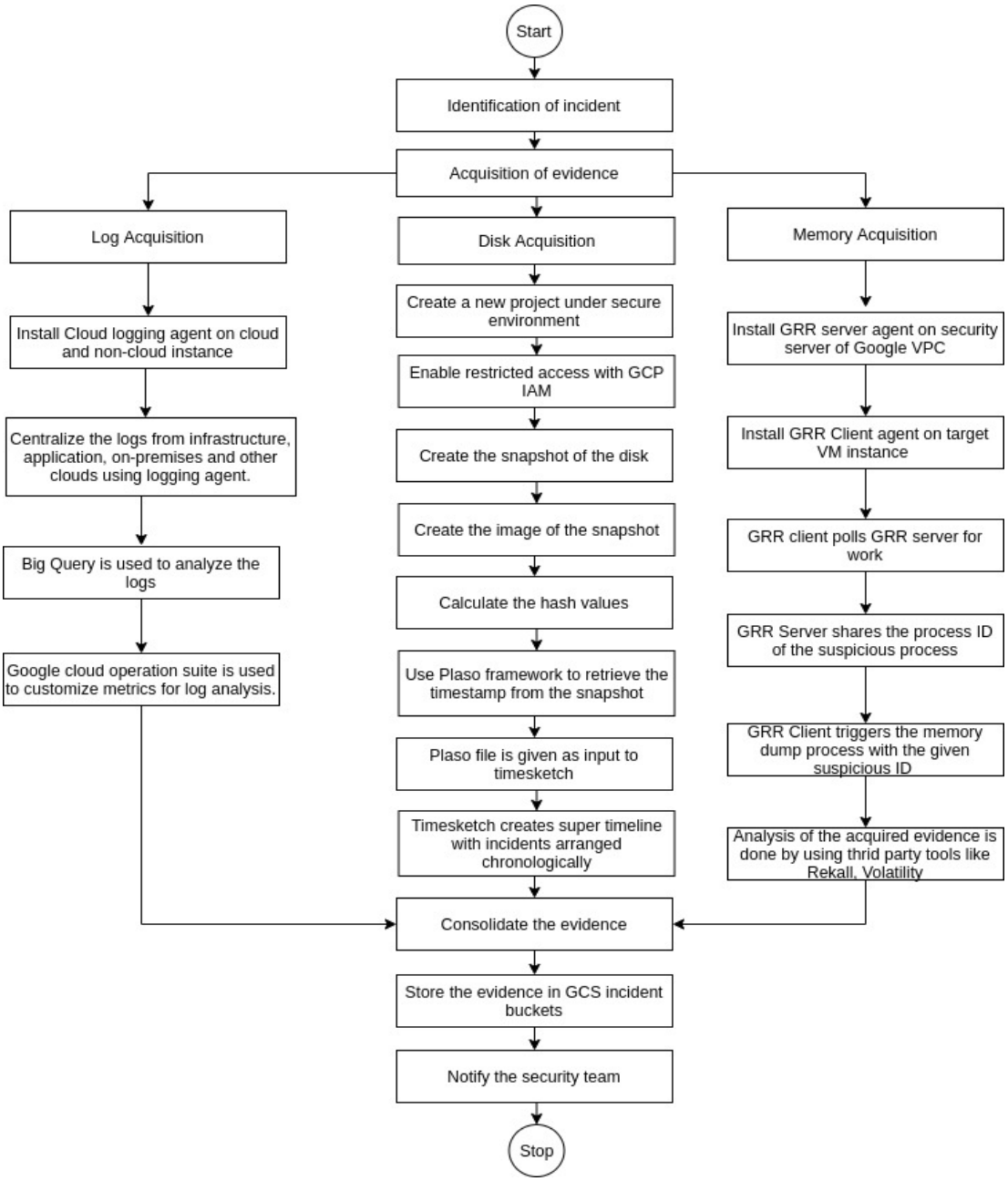


Figure 2.3: GCP Forensics Workflow

2.2.1.2 Evaluating Cloud Forensic Challenges Across Proposed Solutions by Leading CSPs

We now discuss and compare some of the cloud forensic challenges listed in the technical report published by NIST in 2020 [43]. NIST has listed 65 challenges corresponding to cloud forensics in its report. In our survey, we found that none of the leading CSPs are addressing all the 65 challenges. However, we observed that 12 of these 65 challenges (suggested in the NIST technical report) are addressed by these CSPs. These 12 challenges are listed below along with the proposed solutions by Leading CSPs (detailed in table 2.1).

1. Decentralized logs- Collecting cloud logs from distributed cloud resources.
2. Log format unification- Collecting logs and making them convertible to other formats used to centralize them.
3. Lack of transparency- Cloud operational details are not transparent to the user.
4. Data available for a limited time- Data associated with cloud resources are available for a limited time.
5. Isolation of evidence- Isolation of cloud resources from the scalable cloud environment.
6. Imaging the evidence- Imaging large volumes of data.
7. Reconstruction of virtual storage- To create the image from a physical disk.
8. Data integrity and preservation- To preserve data and ensure data integrity for evidence admissibility.
9. Validation of forensic image- Ensure integrity of the image captured for analysis.
10. Authentication and access control- Access control approach and policies in the cloud environment.
11. Reliance on CSP- Dependency on CSP for evidence.
12. Confidentiality and Personally Identifiable Information (PII)- Ensuring the confidentiality of PII upon receiving a legal request for disclosure of information.

Table 2.1: Evaluating Cloud Forensic Challenges across proposed solutions by Leading CSPs

Cloud Forensic Challenge	**Proposed Solution**	**Amazon**	**Azure**	**Google**	**References**
Decentralized logs	Centralize logs form cloud resources	✓	✓	✓	[57] [58] [59] [60]
Log format unification	Convert logs to a standard format	✓	✓	✓	[58] [59] [60]
Lack of transparency	Provide transparency to users	✓	✓	✓	[52] [53] [61]
Data available for a limited time	Data retention policy	✓	✓	✓	[62] [63] [64]
Isolation of evidence	Deregister the instance from scalable cloud resources	✓	X	X	[65]
Imaging the evidence	Scalable storage service	✓	✓	✓	[52] [65] [53] [55]
Reconstruction of virtual storage	Creating snapshot of attached disk volumes	✓	✓	✓	[52] [65] [53] [55]
Data integrity and preservation	Apply retention policies for legal investigation	✓	✓	✓	[62] [63] [64]
Validation of Forensic image	To use hash values of the image	✓	✓	✓	[52] [65] [53] [55]
Authentication and access control	Access control mechanism	✓	✓	✓	[65] [53][66]
Reliance on CSP	Provide LEA access to evidential resources	X	✓	X	[53]
Confidentiality and PII	Disclosure of PII only upon receiving Legal request	✓	✓	✓	[67] [68] [69]

2.2.2 Limitations of Cloud Forensics

The above workflows by the leading CSPs, i.e., Amazon, Azure, and GCP, show that there is a lack of standard operating procedures for conducting forensics in the cloud. Each CSP has its own approach. However, log, disk, and memory forensics are common evidential artifacts considered by the leading CSPs for cloud forensics, as shown in Figure 2.4. Also, one of the major drawbacks noticed in the forensics approach adopted by the leading CSPs is that they have ignored anti-forensics. Thus, in this thesis, we bridge this gap. Figure 2.4 highlights the research direction of this thesis in the context of anti-forensics in comparison to the existing cloud forensic approach adopted by leading CSPs.

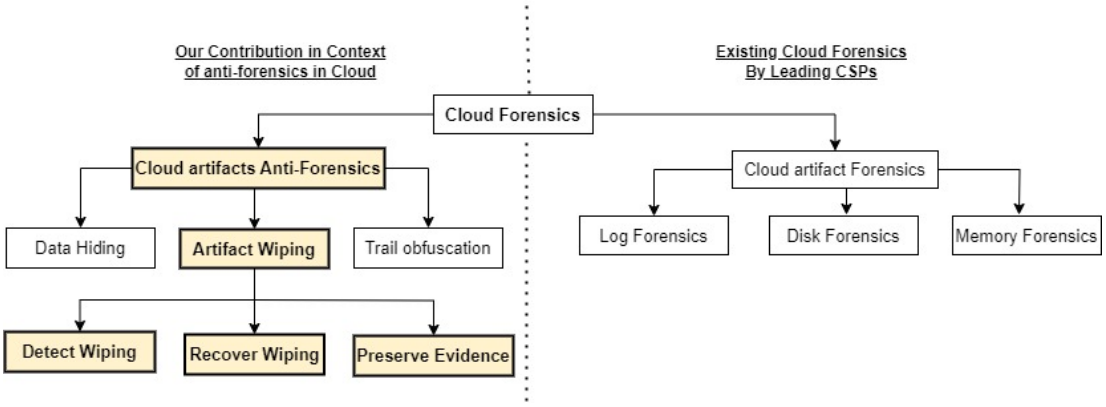


Figure 2.4: Thesis reseach direction in the context of anti-forensics in comparison to the existing systems

2.3 Summary

Adoption of the cloud computing model must inherently support auditing facilities to monitor anti-forensics activities in the cloud, which still doesn't have much-needed attention. We evaluated the forensic procedures implemented by the leading CSPs considering the cloud forensic challenges reported by NIST [43]. Out of 65 challenges, only 12 are addressed, and still, a fair amount of work needs to be done to accomplish other challenges. Anti-forensics is also one of the 65 listed challenges that have not yet been addressed. Moreover, the current forensic strategies of leading CSPs fail to address the associated challenges with anti-forensics. Considering the increased adoption of artifact wiping (an anti-forensic technique) [21], [31], [32], we considered wiping detection, recovery, and preservation in cloud VMs as the main objectives of our thesis.

Chapter 3

Detecting File Wiping

Wiping has a significant effect on file systems. A file system systematically arranges files and directories on the storage media for easy and quick access. Thus, in digital forensics, file systems are accountable for the files stored on digital media and their recovery. Wiping overwrites the contents of the file system data structures (e.g., Master Boot Record (MBR) and Master File Table (MFT) in the NTFS file system, superblock, and inode in the Ext file system) and makes data recovery impossible. The significant application of wiping in organizations is to ensure that they comply with data protection regulations like- a) European Union's GDPR; an individual is given the right to erase their data held with organizations securely). b) United State's HIPPA for by securely erasing Protected Health Information (PHI) from storage media before they are disposed of or repurposed. However, wiping is not just confined to securely deleting the content by authorized users. The adversaries resort to wiping tools to remove the traces of evidence or make data unrecoverable permanently. Wiping attacks are considered the most destructive attacks. Thus, our first objective is to detect file wiping in cloud VMs to ensure the integrity of the evidence. In this chapter, we delve into the intricacies of file wiping detection.

3.1 Challenges in Existing Literature

The forensic tools fail to detect some anti-forensic approaches like artifact wiping and trial obfuscation [70]. The four leading forensic tools AccessData's Forensic Tool Kit (FTK), The Sleuth Kit (TSK), Encase, and OSForensics, could not detect file wiping [70]. Techniques like data carving can help identify wiping, but

it is time-consuming. Also, it cannot determine which specific file was wiped, which is crucial to understand the adversary's intent.

Also, it is observed that most of the earlier works [33], [34], [35], and [36], are detecting file wiping based on the signature of the file wiping tool left behind when the tool is in action. If tool is altered, its corresponding signature also changes, bypassing the security scans. Hence, we need approaches that are independent of tool signatures. As the signatures depend on the hash values, installation traces, deletion traces, etc., it would be challenging to detect wiping as the signatures corresponding to the tools may change over a period of time. Moreover, there are also tools that wipe the traces left behind. Thus, we propose to detect file wiping independent of the traces or signatures left behind by the anti-forensic wiping tools. In this direction, we propose two different approaches. i.e., WiDeJ and WiDeS, to detect wiping.

3.2 Contributions

In this chapter, we detect file wiping in cloud VMs using two different approaches:

- A static approach to detect wiping using file system journals and data blocks. We name this model WiDeJ (**Wi**ping **De**tection using **J**ournals).
- A dynamic approach to detect wiping using system-calls on cloud VM snapshot. We name this model as WiDeS (**Wi**ping **De**tection using **S**ystem-calls)

3.3 Prelimnaries

3.3.1 Data Sanitization

Data sanitization is a specific way in which data overwrites the data on a hard drive or other storage device. There are a number of data sanitization methods that can be used. These methods are also often referred to as data wipe methods, wipe algorithms, and data wipe standards [71].

Most of the wiping tools use these data sanitization methods to overwrite the content in the addressable location multiple times, either with a specific value (i.e., 00/11/AA) or with random characters; this ensures that the original data is

completely unrecoverable. 1-pass overwriting involves overwriting the actual content on the disk for a single time. 2-pass overwriting involves overwriting twice. Likewise, multiple passes involve overwriting the content multiple times (e.g., 3-pass, 7-pass, 35-pass). There are many data wiping standards used by the wiping tools to wipe the content on the storage media. Some of the most adapted standards include- 1) Peter Gutmann, which uses 35-passes to overwrite the content; it uses both random characters and specific characters for wiping; it is the earliest wiping standard. 2) DoD 5220.22-M by United States Department of Defense (DoD) uses 3-passes; it overwrites the contents with 0s in the first pass and 1s in the second pass, and random characters in the third pass. 3) DoD 5220.22-M ECE is another variant of DoD 5220.22-M which uses 7-passes. 4) NIST 800-88 provides media sanitization guidelines based on the storage media using three ways clear, purge, and destroy; of these, the clear technique is the approach used to overwrite the content using multiple passes. 5) HMG Infosec Standard 5 by the government of the United Kingdom has two variants i.e., a) HMG IS5 standard (overwrites using 0s) b) HMG IS5 "Enhanced" Standard (overwrites with 0s followed by 1s. 6) GOST R 50739-95 by Russia uses either 1-pass (overwrite with 0s) or 2-pass (overwrite with 0s followed by random characters) 7)**Random data** is used to overwrite the storage media with random characters. 8) **Write zeros** it is used to overwrite the content with 0s.

3.3.2 Ext File System Journaling

Journals are used to record the changes that are destined for the disk before their execution to ensure file system consistency. In case of system failures, the journal is replayed to restore the file system. The journal entries are grouped as journal transactions. Each transaction begins at the descriptor block and ends with a commit block. Each transaction is further given a sequence number. Only the committed transactions are executed on the disk.

The journal can either capture metadata or both data and metadata corresponding to a file. Ext file system journals come with three journaling modes; based on the content being captured, they are classified as - data journaling, ordered, and writeback modes [72]. Based on the user's requirement, the user can opt for either of these journaling modes.

In data journaling mode, both data and metadata are journaled. This ensures data and metadata consistency with minimal data loss in case of system

failures. This is more reliable for a consistent file system than the other journaling modes. However, this consistency comes with a performance overhead as the write operations are to be repeated twice, i.e., for journal and disk.

In ordered mode, only metadata is journaled; the metadata corresponding to a file is updated in the journal after the data changes are executed on the disk. The data is updated directly at the disk location. In case of system failure, since the metadata changes are captured later the file system consistency is ensured. It is the default journaling mode as it has less overhead and ensures consistency. In write-back mode, only metadata is journaled; unlike in ordered mode the data order is not preserved.

3.3.3 System Calls

We use system-calls in WiDeS to analyze the behavioral characteristics of wiping tools and detect wiping activity. System-call acts as an interface between the user application and the Linux kernel. Based on their functionality and the type of resource system-calls can be classified as process control, file management, device management, information maintenance, communication, and protection. For example, open(), close(), read(), write(), etc., are a few system-calls corresponding to file management. System-calls play a crucial role in malware behavior analysis. In this chapter, we use system-calls to detect wiping activity dynamically.

3.4 Detecting Wiping Using File System Journals and Data Blocks

In the proposed model WiDeJ, we detect file wiping by analyzing the file content at the block level. We use the file system journal to trace the data blocks allocated to the file. Further, we analyze the data blocks and project the difference between the regular file and wiped file. Thus, we detect file wiping independent of the traces or signatures left behind by the anti-forensic tool.

3.4.1 Proposed Model WiDeJ

We propose a model WiDeJ to detect file wiping using file system journals. Here, we shall discuss how to analyze the Ext journal to capture the file's data blocks, followed by the types of wiping and algorithms used to detect file wiping.

3.4.1.1 Ext Journal Analysis

Now, let us analyze the file system journal to trace the file's data blocks. To browse the journal, we used jls command-line tool from TSK. We execute the jls command on the Ext-based VM image. We initialized the image with Ext file system using the mkfs (make file system) command and mounted it for simplicity and a better understanding of the journal. Figure 3.1 shows the journal content just after mounting the file system.

```
(base) idrbt@IDRBT:~/Workspace$ jls dfr-10-ext.dd |more
JBlk    Description
0:      Superblock (seq: 0)
sb version: 4
sb version: 4
sb feature_compat flags 0x00000000
sb feature_incompat flags 0x00000000
sb feature_ro_incompat flags 0x00000000
1:      Allocated Descriptor Block (seq: 2)
2:      Allocated FS Block 67
3:      Allocated Commit Block (seq: 2, sec: 1633839853.2023353600)
```

Figure 3.1: Journal after mounting the file system

The seq: 2 in Figure 3.1 represents the transaction corresponding to file system mounting. Now let us copy a file 'Demo.txt' to the file system and analyze the journal.

As shown in Figure 3.2 we can observe that the sequence number is increasing. Each sequence number represents a single transaction, from line 4 (Allocation of descriptor block) to line 11 (Allocation of commit block) is considered as one complete transaction. This transaction is represented by seq : 3. To analyze the activities performed on the file system, we need to exploit every transaction captured in the journal. The transaction with seq : 3 here corresponds to copying Demo.txt to the file system. Now let us analyze how to access the metadata of this file from the journal transactions.

```
(base) idrbt@IDRBT:~/Workspace$ jls dfr-10-ext.dd |more
JBlk    Description
0:      Superblock (seq: 0)
sb version: 4
sb version: 4
sb feature_compat flags 0x00000000
sb feature_incompat flags 0x00000000
sb feature_ro_incompat flags 0x00000000
1:      Allocated Descriptor Block (seq: 2)
2:      Allocated FS Block 67
3:      Allocated Commit Block (seq: 2, sec: 1633804460.3490855936)
4:      Allocated Descriptor Block (seq: 3)
5:      Allocated FS Block 66
6:      Allocated FS Block 1
7:      Allocated FS Block 67
8:      Allocated FS Block 579
9:      Allocated FS Block 0
10:     Allocated FS Block 65
11:     Allocated Commit Block (seq: 3, sec: 1633804904.2253113088)
12:     Unallocated FS Block Unknown
```

Figure 3.2: Journal after copying a file

We use jcat command-line tool from TSK to interpret the contents for each journal entry from journal blocks. To carve the data from journal block we use command-line utility dd. Figure 3.3 represents the journal block content for inode data of Demo.txt. This entry captures the file metadata such as file size, data block pointers, and other file attributes updated in the file's inode data structure.

The content presented in Figure 3.3 is inode data for the Demo.txt file. We here interpret the data block pointers corresponding to this file. In inode data structure, file data block pointers are from 40 to 87 bytes (direct block pointer), 88 to 91 bytes (single indirect block pointer) 92 to 95 bytes(double indirect block pointers), 95 to 99 bytes (triple indirect block pointers) [73]. By applying this, we have the following block pointers of the file in hexadecimal; 0x0024, 0x0124, 0x0224, 0x0324, 0x0424, 0x0524 (9216, 9217, 9218, 9219, 9220, 9221 are corresponding decimal equivalent values, respectively).

```
(base) idrbt@IDRBT:~/Workspace$ jcat dfr-10-ext.dd 8 7 |dd bs=256 skip=11 count=1|xx
1+0 records in
1+0 records out
256 bytes copied, 0.0300427 s, 8.5 kB/s
00000000: a481 0000 6a59 0000 63e2 6161 63e2 6161  ....jY..c.aac.aa
00000010: 63e2 6161 0000 0000 0000 0100 3000 0000  c.aa........0...
00000020: 0000 0000 0100 0000 0024 0000 0124 0000  .........$...$..
00000030: 0224 0000 0324 0000 0424 0000 0524 0000  .$...$...$...$..
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 4b2a cb1f 0000 0000 0000 0000  ....K*..........
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 2000 0000 040b d82e 040b d82e 040b d82e   ...............
00000090: 63e2 6161 040b d82e 0000 0000 0000 0000  c.aa............
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Figure 3.3: File's inode data captured in journal block

Now that we have the data block pointers for Demo.txt, we analyze its content. For this, we use the blkcat command-line tool from TSK. We pass the image name and the data block pointer as arguments to read the block content. Figure 3.4 shows the content at block pointer 9216.

```
(base) idrbt@IDRBT:~/Workspace$ blkcat dfr-10-ext.dd 9216 |xxd
00000000: 4465 6669 6e69 7469 6f6e 0a41 6e74 692d  Definition.Anti-
00000010: 666f 7265 6e73 6963 7320 6861 7320 6f6e  forensics has on
00000020: 6c79 2072 6563 656e 746c 7920 6265 656e  ly recently been
00000030: 2072 6563 6f67 6e69 7a65 6420 6173 2061   recognized as a
00000040: 206c 6567 6974 696d 6174 6520 6669 656c   legitimate fiel
00000050: 6420 6f66 2073 7475 6479 2e20 5769 7468  d of study. With
00000060: 696e 2074 6869 7320 6669 656c 6420 6f66  in this field of
00000070: 2073 7475 6479 2c20 6e75 6d65 726f 7573   study, numerous
00000080: 2064 6566 696e 6974 696f 6e73 206f 6620   definitions of
00000090: 616e 7469 2d66 6f72 656e 7369 6373 2061  anti-forensics a
000000a0: 626f 756e 642e 204f 6e65 206f 6620 7468  bound. One of th
000000b0: 6520 6d6f 7265 2077 6964 656c 7920 6b6e  e more widely kn
```

Figure 3.4: Block content of a file at specific block pointer (i.e 9216)

We determine the block content byte-by-byte at each block allocated to Demo.txt. The block size of Ext file system is 4096 bytes. Hence, we determined that 4096 characters were present in each block and repeated the same for all blocks allocated to the file. The file content here will be in hexadecimal format. We analyze this file content to detect file wiping. Algorithm 1 details the procedure for analyzing the journal, capturing the file content from the file's data blocks, and detecting file wiping.

Algorithm 1 Analyze journal for file data blocks and detect file wiping

```
procedure AnalyseJournal(VM_img)
    JrnlFile ← getJrnl(VM_img)
    JrnlTranSeqNum[] ← getSeqNum(JrnlFile)
    for all JrnlTranSeqNum do
        jrnlEntry ← getInodeEntry()
        dataBlkAddr[] ← getBlkPointer(jrnlEntry)
        for all dataBlkAddr do
            hexFile ← readBlkData()
        Dict ←UpdateDictionary(hexFile)
        entropy ←ComputeEntropy(Dict)
        if entropy == 0 then
            print file wiped with specific character
        else if entropy == 0.99 then
            check file header to detect wiping
            if header has random characters then
                print file wiped with random characters
    return
```

3.4.1.2 Using Shanon's Entropy

Every character from data blocks in hexadecimal format is converted to ASCII (American Standard Code for Information Interchange) equivalent decimal value, i.e., between 0 and 255. We create a dictionary to store these ASCII characters. The dictionary's key stores the ASCII values from 0 to 255, and the corresponding values store each character's cumulative sum of their occurrence in all blocks allocated to the file. Algorithm 2 details the procedure for the same. We further compute each character's probability of occurrence and the entropy for the entire file. Based on the file entropy obtained, we determine file wiping.

Algorithm 2 Initialize the dictionary to store the ASCII values and frequencies

```
procedure UpdateDictionary(hexFile)
    Dictionary Dict{ascii, frequency}
    while !EOF(hexFile) do
        char ← readChar(hexFile)
        ascii ← covertToASCII(char)
        frequency ← Dict.get(ascii)
        if frequency == 0 then
            frequency ← frequency + 1
            Dict.add(key, frequency)
        else
            Dict.add(key, frequency + 1)
    return Dict
```

Shannon's entropy is a good metric for measuring the randomness in characters [74]. Shanon was the first person to propose a measure of uncertainty or randomness of probability distribution and termed it as "entropy" [75]. We use Shanon's entropy to detect file wiping in the proposed model. We further normalize the computed entropy value as normalized measures of entropy are much closer to one another when compared with absolute entropy. The Shannon's entropy is shown in Equation 3.5, and normalized Shannons entropy is shown in Equation 3.2. Here, E is the Shanon's entropy, and NE is Normalized Entropy, X is the ASCII character, and $P(X)$ is the probability of the character X in the given file.

$$E = -\sum_{i=1}^{N} p(X_i) \log_2 p(X_i) \tag{3.1}$$

$$NE = -\frac{1}{\log_2 N} \sum_{i=1}^{N} p(X_i) \log_2 p(X_i) \tag{3.2}$$

We computed probabilities for each ASCII character of the file stored in the dictionary. Later, we computed the entropy of the file using the formula shown in Equation 3.1 and normalized the entropy using the formula shown in Equation 3.2. The value of normalized entropy lies between 0 and 1 and is used to detect file wiping. Algorithm 3 details the procedure to compute entropy.

Algorithm 3 Compute file entropy

```
procedure ComputeEntropy(Dict)
    for all key ∈ Dict do
        totalCharCount ← totalCharCount + Dict.getValue(key)
    for all key ∈ Dict do
        charProbability ← Dict.getValue(key)/totalCharCount
        entropy ← entropy + charProbability * log(charProbability)
    N ← 256  ▷ N is assigned with total ASCII characters 256 (.i.e 0 to 255)
    entropy ← entropy/log(N)
    return entropy
```

We now wipe the contents of the file and interpret the file content. We experimented the proposed model with tools, i.e., Shred, and Wipe. By default, these tools wipe the file with random characters unless the user explicitly specifies to wipe the file with a specific character like 0 or 1.

The wiping tools overwrite the file either with random characters or with specific characters like 0s or 1s [70]. Thus, we have designed two cases based on how a file can be wiped.

- Case 1: Wiping a file with a specific character.
- Case 2: Wiping a file with random characters.

Case 1: Wiping a file with specific character

Wiping a file with 0s or 1s overwrites the content of regular readable data with either 0 or 1. To perform file wiping with 0s, we used Shred, a command-line tool that securely deletes files by wiping them. After wiping the file, we analyzed the journal entries again and traced the block pointers. Figure 3.5 shows the journal entries after wiping the file. Here we are interested in journal entries 13, 16, and 19. Likewise, we edit the jcat command with corresponding journal entries (i.e., 13, 16, and 19) to view the file's data block pointers.

```
(base) idrbt@IDRBT:~/Workspace$ jls dfr-10-ext.dd |more
JBlk    Description
0:      Superblock (seq: 0)
sb version: 4
sb version: 4
sb feature_compat flags 0x00000000
sb feature_incompat flags 0x00000000
sb feature_ro_incompat flags 0x00000000
1:      Allocated Descriptor Block (seq: 2)
2:      Allocated FS Block 67
3:      Allocated Commit Block (seq: 2, sec: 1633839853.2023353600)
4:      Allocated Descriptor Block (seq: 3)
5:      Allocated FS Block 66
6:      Allocated FS Block 1
7:      Allocated FS Block 67
8:      Allocated FS Block 579
9:      Allocated FS Block 0
10:     Allocated FS Block 65
11:     Allocated Commit Block (seq: 3, sec: 1633839868.3453514496)
12:     Allocated Descriptor Block (seq: 4)
13:     Allocated FS Block 67
14:     Allocated Commit Block (seq: 4, sec: 1633839891.3510770176)
15:     Allocated Descriptor Block (seq: 5)
16:     Allocated FS Block 67
17:     Allocated Commit Block (seq: 5, sec: 1633839892.41928704)
18:     Allocated Descriptor Block (seq: 6)
19:     Allocated FS Block 67
20:     Allocated Commit Block (seq: 6, sec: 1633839898.1270903040)
21:     Unallocated FS Block Unknown
```

Figure 3.5: Journal after wiping a file

Further, upon examining the block content by using blkcat command for journal entries 13, 16, and 19, we found that the entire file content was wiped with 0s. Figure 3.6 shows the block content of the file wiped with 0s at block address 9216. An important observation from Figure 3.6 is that the frequency of character 0 is equivalent to file size, and the frequency of all other characters is 0. This is because all the characters in the file are overwritten with a specific character '0'. Thus, we detect file wiping using journals and Shanon's entropy by applying Algorithm 1, 2 and 3.

```
(base) idrbt@IDRBT:~/Workspace$ blkcat dfr-10-ext.dd 9216 |xxd |more
00000000: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Figure 3.6: Block content after wiping a file with zero

Case 2: Wiping a file with random characters

By default, the wiping tools wipe the file with random characters. Figure 3.7 shows the block content of the file when wiped with random characters. Detecting file wiping when wiped with random characters is more challenging than detecting file wiping when wiped with specific characters. The reason for the same is discussed in Section 3.4.2.

```
(base) idrbt@IDRBT:~/Workspace$ blkcat dfr-10-ext.dd 9216 |xxd |more
00000000: 8aa0 4a6e d26f d37f 2382 988f 99a5 33fa  ..Jn.o..#.....3.
00000010: 267d 11c3 99b0 090b 3bcb be98 2551 fd4f  &}......;...%Q.O
00000020: 7fe1 ffc7 905c f06f 4c3d 8fb5 c49b 6a56  .....\.oL=....jV
00000030: 5110 6b96 294f 93f3 c43c e38c b8ef 726a  Q.k.)O...<....rj
00000040: 0df9 e09e ee5d c332 5e5a 7b0c d89d b01b  .....].2^Z{.....
00000050: 82e6 4714 362b 64a9 8ec9 dd99 2da8 a2ef  ..G.6+d.....-...
00000060: 0cb9 cbde c6ae 24ad 434d 281c 1016 dfe5  ......$.CM(.....
00000070: 3faa 89e5 f893 9198 cf7f 7972 1d0b cef9  ?.........yr....
00000080: 9c16 c61d 22a5 e41f 18b6 6674 6163 f9a4  ....".....ftac..
00000090: 074e ac9a ca54 7328 05ae 093f 3716 09a4  .N...Ts(...?7...
000000a0: 8ad4 98f1 2e06 2772 997d 1433 ba8e 152b  ......'r.}.3...+
000000b0: 2f91 2147 ae4a c7d9 49ca 5713 d4c0 258f  /.!G.J..I.W...%.
```

Figure 3.7: Content at data block when wiped with random characters

3.4.2 Results and Discussion

In this section, we present the results of our model WiDeJ. We initially evaluate the scope of recovering the journal entries corresponding to a file when the files are wiped. We cannot fetch the data blocks corresponding to a wiped file if the journal entries are overwritten. Thus, it becomes crucial to determine the scope of recovering journal entries corresponding to a wiped file. This depends on two major factors, i.e., the wiping tool and file size; to support this statement, we have executed an experiment with a cloud VM image whose size is 1 GB and its journal size is 8 MB. We initialize the image to a clean state using mkfs command. Later, we used two different wiping tools, Shred and Wipe alternatively, to wipe the file by varying file size from 1 MB to 30 MB. For each file, after wiping it, we checked for the journal entries to see if they were overwritten. Figure 3.8 shows the result of this experiment.

From Figure 3.8, we observe that the scope of recovering the journal entries for wiped files decreases as the file size increases; smaller file sizes result in fewer journal entries (less no.of write operations) when wiped, while larger file sizes lead to more journal entries and potentially overwriting early journal entries. Also, we notice that we have more scope of recovery when a file is wiped with a Shred tool compared to a Wipe tool. This is because the Shred tool uses a trivial data

wiping standard i.e., DoD 5220.22-M, compared to the Wipe tool that uses Peter Gutmann's algorithm. The increased number of passes in the Wipe tool results in more write operations, leading to increased journal entries. As the journal entries increase, there may be a chance that the earlier journal entries may get overwritten. The results in Figure 3.8 show that we can recover the journal entries corresponding to a file whose size is less than 30 MB when wiped using basic wiping tools like Shred, but we can recover a file with its size less than 1 MB when wiped with a rigorous wiping tool (like Wipe tool).

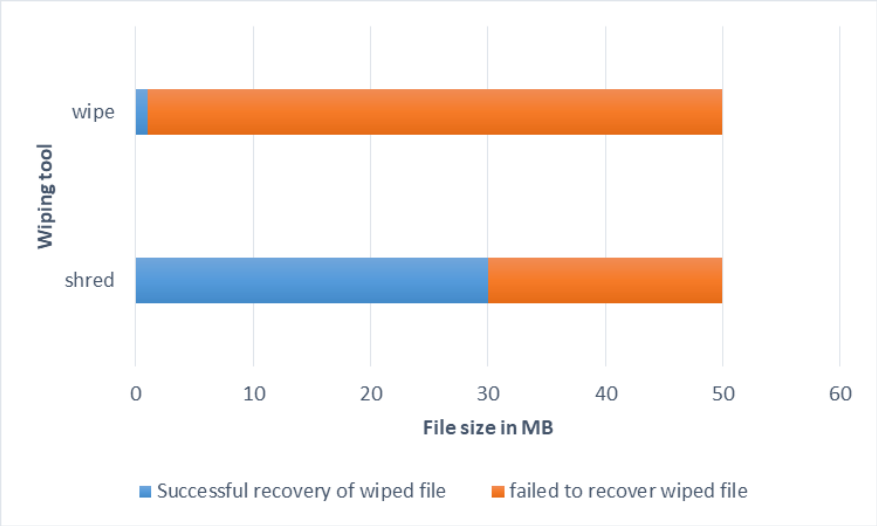


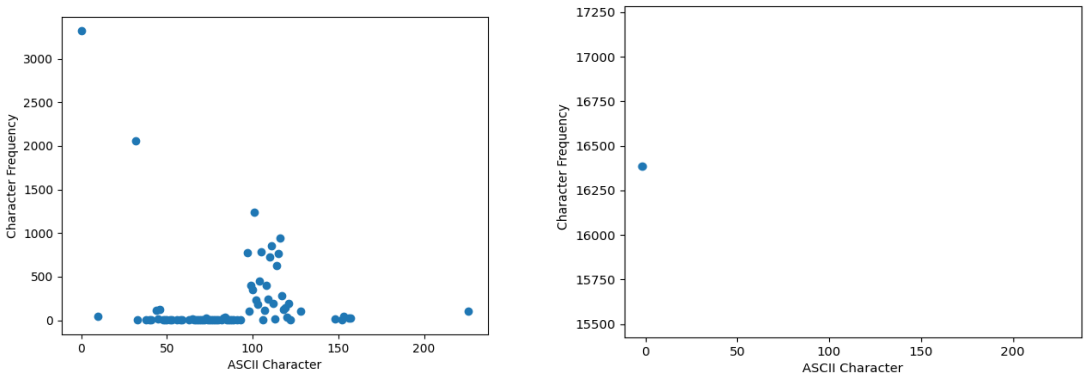
Figure 3.8: Scope of recovering wiped file data blocks using file system journals by varying wiping tools and file size.

Let us assume that we could fetch the data blocks allocated to the file using the journals. We now compute the entropy of characters on these data blocks. We compute the file's entropy using Shanon's entropy using the Equations 3.1, and 3.2. We considered 54 files, which included 20 different file types. Table 3.1 shows the entropy value for different file types before and after wiping with random characters.

Table 3.1: Entropy values for different file types

File Type	Entropy before wiping	Entropy after wiping	File type	Entropy before wiping	Entropy after wiping
.dat	0.83	0.99	.pptx	0.99	0.99
.log	0.31	0.99	.eps	0.51	0.99
.png	0.94	0.99	.exe	0.99	0.99
.bmp	0.98	0.99	.obj	0.58	0.99
.jpg	0.99	0.99	.aspx	0.96	0.99
.pdf	0.99	0.99	.mp3	0.98	0.99
.eml	0.62	0.99	.mp4	0.84	0.99
.docx	0.86	0.99	.zip	0.99	0.99
.xlsx	0.94	0.99	.odt	0.93	0.99
.ods	0.99	0.99	.odp	0.99	0.99

For case 1, where a file is wiped with specific characters like 0s and 1s the computed file entropy is equivalent to 0. Figure 3.9a and 3.9b show the character distribution in a file Demo.txt before and after wiping. Here, we can see that the frequency of characters is unevenly distributed before wiping the file Demo.txt, but after wiping, the frequency of the character '0' is equivalent to the file size on the disk. In this case, for Demo.txt, the frequency of the character 0 is 16384 (i.e., file size is 16 KB), and the probability of occurrence of ASCII character '0' is 1; upon substituting the probability in Equation 3.1 and 3.2, the entropy is 0. Thus, if the entropy of a file is 0, we conclude that the file has been wiped with a specific character.



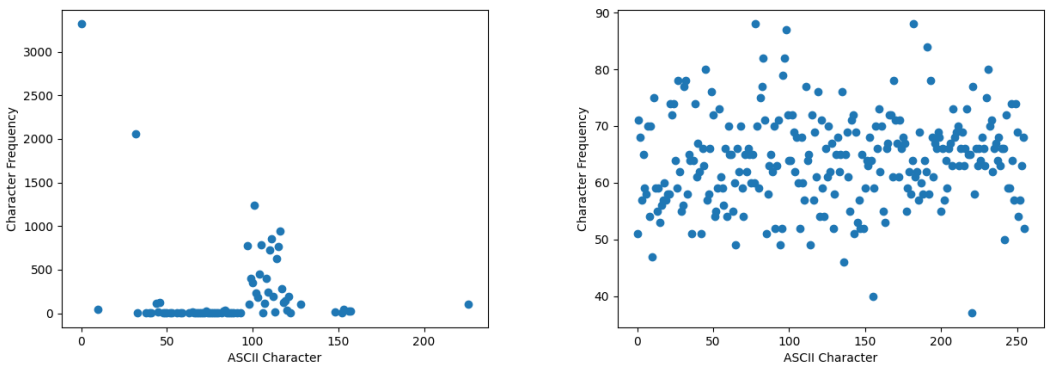
(a) Frequency of ASCII characters in regular file

(b) Frequency of ASCII characters in file after wiping with 0

Figure 3.9: Difference between regular file and file wiped with 0s

For case 2, detecting file wiping when a file is wiped with random characters is

quite challenging as the entropy value of regular files, e.g., .jpg, .pdf, .mp4, .mp3, and other file formats would be close to 1 (i.e., 0.96, 0.97, 0.98, and 0.99), as shown in the Table 3.1, and the entropy values when the file wiped with random characters is 0.99; making it difficult to differentiate between regular files and files wiped with random characters. Figure 3.10a, and 3.10b shows the character distribution before file wiping and after wiping a file with random characters. We can see that the distribution of characters is quite evenly distributed across 256 ASCII characters in Figure 3.10b. More randomness leads to higher entropy because entropy is the average of the logarithmic function of the probability i.e., when $p(X_i)$ is small, $\log_2 p(X_i)$ is large. Thus, when a file is wiped with random characters, the file entropy is high, i.e., 0.99.



(a) Frequency of ASCII characters in regular file

(b) Frequency of ASCII characters after wiping with random characters

Figure 3.10: Difference between regular file and file wiped with random characters

Based on the observed values from Table 3.1, we notice that when a file is wiped with random characters, the entropy is 0.99. Thus, we fixed the threshold at 0.99 to detect wiping. We classify the files with an entropy value of 0.99 as file wiping. However, to reduce false positives, we further check the file header and classify the file as a regular or wiped file.

File header can play a crucial role in determining file wiping. Figure 3.11 taken from paper [1] shows file structure layout. The initial few bytes of the file constitute the file header. It contains file type, file size, and other metadata corresponding to a file. If a file is wiped with random characters, the header and footer are overwritten with random characters. On the other hand, for regular files (e.g., .jpg, .pdf, .ppt, etc.), though the file data has random characters and

an entropy of 0.99, they have a well-defined header structure that can be used to determine the file type and fetch metadata corresponding to the file. However, there are chances that the adversary may tamper with the file header, a basic anti-forensic approach adapted to hide the file type and mislead the investigator. Thus, we consider both file entropy and file header to detect wiping. If the file's entropy is 0.99 and even the header contains random characters, we classify the file as a file wiped with random characters.

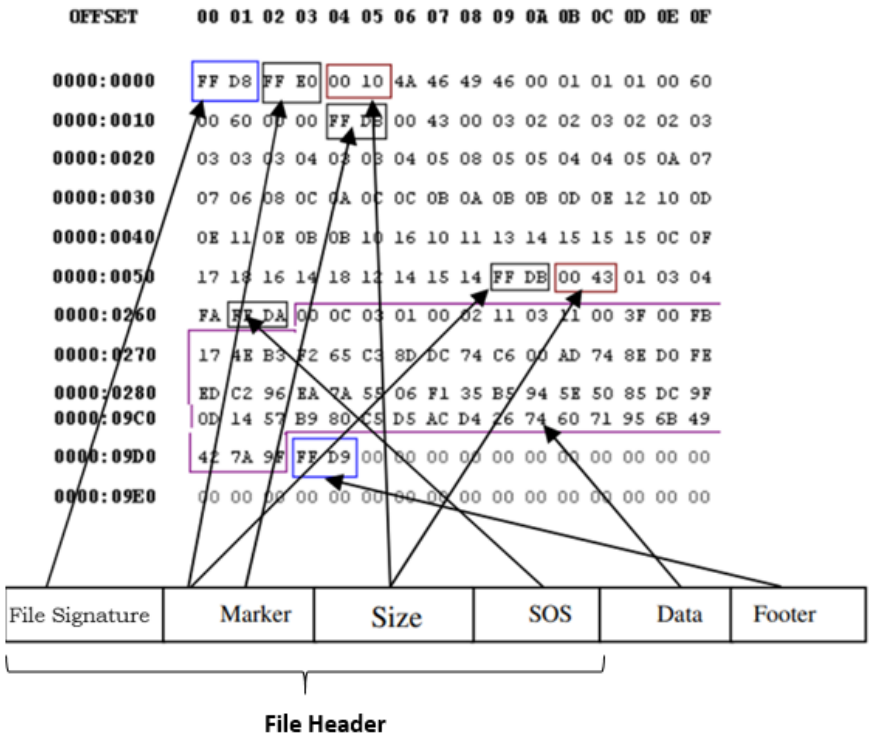


Figure 3.11: File layout (reproduced from [1])

We also compared our model WiDeJ with the approach by Savoldi et al. [2], which is closely related to our work. Savoldi et al. used data blocks and a statistical test suite developed by NIST to determine the randomness of block content and detect wiped data fragments on disk in comparison to standard data fragments. Table 3.2 highlights the differences between WiDeJ and their approach. However, the objectives of WiDeJ and the method by Savoldi et al. differ. WiDeJ detects wiping at the file level, identifying which specific file has been wiped, which helps investigators understand the adversary's intent. In contrast, Savoldi et al.'s

method identifies wiped data fragments on disk. Therefore, while their approach indicates that wiping was used, it does not specify which file was wiped.

Table 3.2: Comparison of WiDeJ and [2]

Existing approach [2]	**WiDeJ**
1. Disk level wiping detection.	1. File-level wiping detection.
2. Uses NIST statistical tests to detect wiping.	2. Used file system journals and entropy to detect wiping.
3. Considered 15 file types.	3. Considered 22 file types.
4. Use an additional 5 statistical tests to reduce false positives.	4. Use file header to reduce false positives.
5. Detection time is prolonged as five statistical tests are repeated on the whole disk.	5. Detection time is less as we consider only the files listed in the journal.

Finally, we could detect file wiping using our model WiDeJ. For case 1, we were able to detect file wiping with 100% accuracy. For Case 2, based on entropy, we were able to detect file wiping with 90% accuracy (here, we computed the accuracy using the equation 3.9 discussed in section 3.5.4) on our dataset of 54 files. Further, we reduced the false positives and detected file wiping with 100% accuracy by considering file's header. However, WiDeJ fails to detect wiping in two scenarios i.e., 1) if the journal entries corresponding to a wiped file are overwritten, 2) if the file header is tampered. Thus, we need an approach which is independent of these dependencies. Hence, we propose our next approach WiDeS, where we detect file wiping using system-calls.

3.5 Detecting Wiping using System-calls

An approach that detects an anti-forensic wiping tool based on its behavior rather than relying on its signature can be more robust and reliable. Thus, in the proposed model, WiDeS, we use Sequence of System-calls (SoS) and information theory metrics to analyze the behavior of wiping tools and detect wiping attacks.

3.5.1 Role of System-calls in Behavior Analysis

Early study shows that system-calls played a significant role in classifying normal process behavior from malicious process behavior [14]. The author in the paper [16] builds a database of a short sequence of system-calls and compares every new sequence of system-calls with these patterns. The distance between them determines the deviation between normal and abnormal behavior. Contrary to work proposed in the paper [16], the author in paper [76] proposes maintaining a dictionary for anomalous patterns using the sequence of system-calls and determining if these patterns of the system-calls are normal or abnormal. The author in [15] proposed a solution for an anti-detection feature the malware uses, i.e., system-call injection attack. The author used the information theory property Asymptotic Equipartition Property (AEP) to extract system-calls rich in information for malware detection. The authors in [77] proposed ShieldFS, an approach to detect ransomware attacks and revert their effects on Windows machines using I/O Request Packets (IRPs). To detect ransomware attacks, they used features like frequency of read and write operations, write entropy, file renamed, files accessed, and folder listing.

3.5.2 Proposed Model WiDeS

The proposed model, WiDeS, detects wiping by carefully profiling the benign process behavior and the process behavior perturbed by wiping. We monitor the system behavior at the process level by using short patterns from SoS invoked by these processes. WiDeS contains three modules, as listed below; the output of each module is the input for the next module.

1. **Profiling Process behavior:** In the first module, we build a short patterns using SoS corresponding to a process. Based on these patterns and their frequencies, we compute the entropy for each process and classify the process as a wiping or benign process. There may be cases where the benign process may be falsely classified as a wiping; thus, the processes determined as wiping are sent to the next module for further classification.

2. **Filtering driven by write() system-call:** In the second module, we get the pattern with maximum frequency for each process classified as wiping. We check if this pattern contains the write() system-call. If the pattern contains a write() system-call, we consider it wiping, as it involves overwriting

content multiple times. Conversely, we classify the process as benign if the pattern does not contain a write() system-call.

3. **Analysis of buffer data:** We further improve the classification in the third module by considering the buffer data pulled from write() system-call parameters. Buffer data is the content stored in the buffer as received from an input device or other process; later, this content is pushed to the file using the write() system-call. In wiping, we know that the file content is wiped with specific characters, random characters, or repeated patterns of characters; thus, we determine wiping based on buffer content.

Algorithm 4 gives the abstract view of WiDeS. A detailed explanation with examples is discussed in the subsequent sections.

Algorithm 4 WiDeS Algorithm

```
procedure WiDeS(processes[P_i])
    for all processes[P_i] do
        patterns[] ← CreatePatterns(SoS(P_i))
        Dict{ptn, freq} ← CreateDict(patterns)
        Entropy ← ComputeEntropy(Dict)
        if Entropy <= Threshold then                          ▷ Module 1
            ptn ← getPtnWithMaxFreq(Dict)
            if write() ∈ ptn then                             ▷ Module 2
                buff[] ← readBufferData(write)
                if buff[] ∈ {specific, random, pattern} then  ▷ Module 3
                    Classify as wiping process
    return
```

3.5.2.1 Profiling Process Behaviour

A. Collect SoS and build patterns

We profile the process behavior using its SoS. To collect the SoS, we used the Sysdig tool. Sysdig tool is an open-source tool used to perform system monitoring [78]. We deployed the Sysdig tool on the target virtual machine to gather the SoS associated with each process. Using process IDs, we obtained the SoS for all processes within the given time frame. In WiDeS, we profiled benign activity using

79 randomly selected user processes and wiping activity using 5 different wiping tools (Shred, SRM, Scrub, Sfill, and Wipe). We collected the SoS corresponding to all these processes. Further, we build patterns with short SoS to profile process behavior.

Early literature suggests that short SoS are good classifiers that distinguish between normal activity and abnormal activity [16]. Thus, we have opted for short patterns from the system-call sequence as observable discriminators. These patterns include three consecutive system-calls. We construct patterns composed of three consecutive system-calls for each given process, denoted as P_i, where i ranges from 1 to N, N representing the total number of processes within the given time frame. We collect the corresponding SoS as $SoS(P_i) = s_1, s_2, s_3, ..., s_n$, where each s represents a system-call (e.g., open(), read(), write(), etc.). We consider the system-calls s_j to build patterns, where j $\in$ 1 to n. Here, $s_{j=1}$ denotes the first system-call, s_{j+1} denotes the next consecutive system-call, and $s_{j=n}$ denotes the last system-call corresponding to each process. We create pattern ptn, using three consecutive system-calls, i.e., ptn=s_j, s_{j+1}, s_{j+2}. By iterating over the values of j from 1 to n we get k no.of patterns (here, $k = n-2$) as shown below,

$$
\begin{aligned}
ptn_1 &= s_1 + s_2 + s_3 \\
ptn_2 &= s_2 + s_3 + s_4 \\
ptn_3 &= s_3 + s_4 + s_5 \\
&\vdots \\
ptn_k &= s_k + s_{k+1} + s_{k+2}
\end{aligned}
$$

Algorithm 5 details the steps in creating patterns. The CreatePatterns method accepts the input as $SoS(P_i)$, here i $\in$ 1 to N, where N is the total number of processes executed in the given time frame and P_i denotes the i^{th} process. We create an array for every process and store these patterns.

Algorithm 5 Create patterns from SoS for each process

```
procedure CreatePatterns(SoS(P_i))
    patterns[]
    for all j = 1 to n − 2 do
        patterns[j] ← s_j + s_{j+1} + s_{j+2}
    return patterns
```

B. Profiling Benign Process Behavior

In profiling benign process behavior, we collect the SoS of 79 randomly selected user processes (e.g., vi, ls, cp, pwd, etc.) and build a repository of these system-calls. We now create an array of patterns corresponding to each process, using Algorithm 5. Example 1 elaborates the pattern construction for a benign process 'ls'.

Example 1. *Process 'ls' lists the files and sub-directories in the current directory. We collect SoS for the process 'ls' using the Sysdig tool. Figure 3.12 shows the subset of SoS for this particular process (for better readability, we listed only a few initial system-calls).*

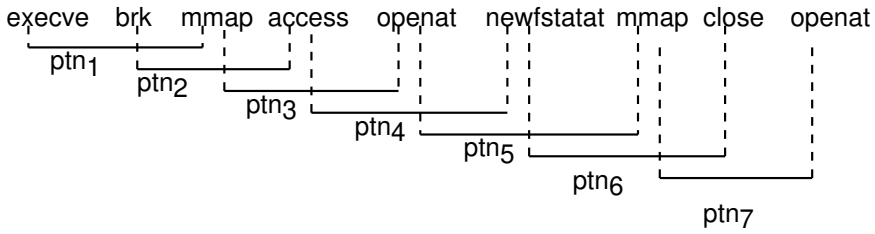


Figure 3.12: Subset of SoS for benign process 'ls'

We now take a window of size 3 to capture 3 consecutive system-calls to create 1 pattern. Likewise, we slide the window across the SoS and create multiple patterns, as shown in Figure 3.12. We now have 7 unique patterns as shown in Figure 3.13.

ptn_1: execve brk mmap

ptn_2: brk mmap access

ptn_3: mmap access openat

ptn_4: access openat newfstatat

ptn_5: openat newfstatat mmap

ptn_6: newfstatat mmap close

ptn_7: mmap close openat

Figure 3.13: Unique patterns of benign process 'ls'

C. Profiling Wiping Process Behavior

Similar to benign process profiling, we profile the behavior of the wiping process using 5 different wiping tools (Shred, SRM, Scrub, Sfill, and Wipe). Example 2 shows pattern construction for a wiping process 'Shred'.

Example 2. *Shred is a common file-wiping process used to delete a file securely. We collect SoS for the Shred process. For the convenience of reading, we present the subset of SoS for the Shred process in Figure 3.14. Here, we use a sliding window of size 3 to capture patterns as shown in Figure 3.14.*

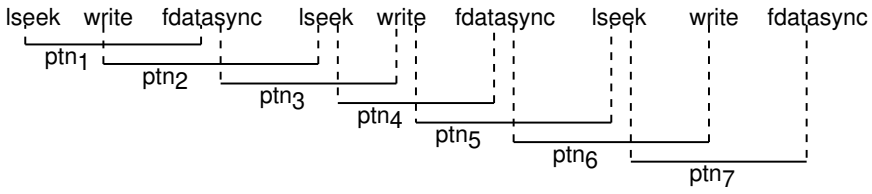


Figure 3.14: Subset of SoS for wiping process 'Shred'

Figure 3.15 shows the unique pattern obtained from the Shred process. Unlike in the case of the benign process 'ls' shown in Example 1 where we get 7 unique patterns, here we get only 3 unique patterns. From Figure 3.14 we see that ptn_4, ptn_5, ptn_6 *and* ptn_7 *are similar to* ptn_1, ptn_2, *and* ptn_3, *we get only three unique patterns.*

ptn_1: lseek write fdatasync
ptn_2: write fdatasync lseek
ptn_3: fdatasync lseek write

Figure 3.15: Unique patterns of wiping process 'Shred'

Using these patterns as a basis, we found an interesting characteristic that set apart benign and wiping processes, i.e., randomness in patterns. Using Example 1 and 2, we can determine that the randomness of patterns is higher in the benign process than in the wiping process. Hence, we profile the behavior of processes by carefully examining the degree of randomness in the generated patterns. We store these patterns and their frequencies in a dictionary.

Algorithm 6 Build the dictionary to store the patterns and their frequencies

```
procedure CreateDict(patterns[])
    Dict{ptn, freq}
    for all patterns do
        ptn ← getPtn(patterns)
        freq ← Dict.get(ptn)
        if frequency = 0 then
            freq ← freq + 1
            Dict.add(ptn, freq)
        else
            Dict.add(ptn, freq +1)
    return Dict
```

Algorithm 6 builds a dictionary $Dict\{ptn, freq\}$ for each process to map the patterns with their frequencies. In Dict, the pattern ptn is the key, and the frequencies are the values. We initially check if the pattern exists in the dictionary; if so, we increment the frequency of the corresponding pattern. If the pattern is not yet present, we add it to the dictionary with its frequency as 1. We built the dictionary for all 79 benign processes and 5 wiping processes.

Table 3.3, Table 3.4 lists the patterns and their frequencies for benign and wiping processes, respectively (we included only a few patterns and their frequencies in the tables due to space constraints). Table 3.3, Table 3.4 shows the difference between the benign and wiping processes' patterns frequencies, i.e., in wiping processes, the patterns' frequency is much higher than benign processes.

In the context of detecting wiping, we observe how often a pattern is repeated in a process. Processes with a high frequency of patterns indicate characteristic behavior associated with the wiping process's activity. As wiping involves overwriting the contents multiple times, the same system-call patterns are invoked multiple times, increasing the probability of occurrence of patterns in the wiping process. Conversely, processes that exhibit infrequent patterns that reflect diverse or random patterns with less probability are often found in benign processes. Thus, we can classify the process based on the degree of randomness of patterns. In WiDeS, we use Shanons' entropy to measure the randomness of patterns.

Table 3.3: Benign processes' patterns and their frequencies

Process	Patterns	Frequency of patterns
Touch	archprctl access openat	1
	access openat fstat	1
	openat fstat mmap	2
	fstat mmap close	2
cut	brk archprctl access	1
	archprctl access openat	1
	openat fstat mmap	2
	fstat mmap close	2
rm	openat read pread64	1
	read pread64 pread64	1
	pread64 pread64 pread64	2
cmp	pread64 pread64 pread64	2
	pread64 pread64 fstat	1
	pread64 fstat mmap	1
echo	pread64 pread64 pread64	2
	pread64 pread64 fstat	1
	pread64 fstat mmap	1

Table 3.4: Wiping processes' patterns and their frequencies

Process	Patterns	Frequency of Patterns
Shred	fcntl lseek write	35
	lseek write write	35
	write write write	3584
	write write fdatasync	35
	write fdatasync fcntl	34
	fdatasync fcntl lseek	34
SRM	lseek write write	38
	write write write	15428
	write write fsync	38
	write fsync lseek	37
	fsync lseek write	37
Wipe	write lseek write	35
	lseek write write	35
	write write write	16623
	write write fdatasync	35
	write fdatasync write	34
	fdatasync write lseek	34
scrub	lstat stat openat	35
	write write write	1666
	write write fsync	35
	write fsync fadvise64	35
	fsync fadvise64 close	35
	fadvise64 close write	34
	close write ioctl	33
sfill	write write write	1002928

D. Computing Entropy

Shannon entropy is one of the well-known approaches to measure randomness or uncertainty in the given data. In digital forensics, entropies are commonly used to determine the file type disguised as other file types [79]. In earlier works [2], [74], Shannon entropy was used to detect wiping. In paper [2], entropy values were used to detect disk fragments wiped with random characters. In paper [74], they detect file wiping using machine learning algorithms. They used the entropy value of the file name overwritten multiple times with random characters as one of the features in their model to detect wiping. In paper [77], the entropy of the write operation is used to determine ransomware attacks. In WiDeS, we use the entropy of the patterns to determine wiping. We used entropy values to compute the probability distribution of patterns created using SoS corresponding to a process.

To determine the probability of a pattern in a process, we compute the ratio of the frequency of a pattern to the sum of frequencies of all patterns (S) in a process. We compute S using the Equation 3.3, here, $p(ptn_i)$ is the probability of occurrence of the i^{th} pattern, and k is the total no.of patterns in a process. We compute the probability of the pattern using the Equation 3.4.

$$S = \sum_{i=1}^{k} frequency(ptn_i) \tag{3.3}$$

$$p(ptn_i) = \frac{frequency(ptn_i)}{S} \tag{3.4}$$

The Shannon entropy for the process is denoted as E_P and is computed using Equation 3.5. We further normalize the entropies between 0 and 1, allowing us to compare the entropy of different processes, even if they have different possible outcomes. We compute NE for each Process P denoted as NE_P using the Equation 3.6.

$$E_P = -\sum_{i=1}^{k} p(ptn_i) \log_2 p(ptn_i) \tag{3.5}$$

$$NE_P = \frac{E_P}{\log_2 S} \tag{3.6}$$

Algorithm 7 details the steps for computing the entropy values in WiDeS. The computed entropy values for the wiping and benign processes are shown in Table 3.5. Due to space constraints and improved readability, we have included entropy values of a few benign processes. This table gives us fundamental insight into how the entropy values of the wiping and benign processes differ. It is observed that the entropy values for wiping processes are less in comparison to the entropy values of the benign processes. This difference between the entropies is due to the random occurrence of patterns. In the wiping process, the patterns are more frequent and less random; conversely, the patterns are more random and less frequent in benign processes. Hence, benign processes have higher entropies than wiping processes.

Algorithm 7 Compute entropy of each process

```
procedure ComputeEntropy(Dict{ptn,freq})
    S ← 0
    for all ptn in Dict do
        S ← S + Dict.getPtnFreq(ptn)
    for all ptn in Dict do
                                ▷ Here 'P' is the process and 'p' is the probability
        p ← Dict.getPtnFreq(ptn)/S
        E_P ← E_P + p * log(p)
        NE_P ← E_P/log(S)
    return NE_P
```

Table 3.5: NEs of wiping and benign processes

Wiping Process	NE	Benign Process	NE
Shred	0.42	touch	0.95
SRM	0.011	cut	0.92
Scrub	0.44	rm	0.93
Wipe	0.018	echo	0.94
Sfill	0.000077	cmp	0.95

We need to determine the Threshold (Th) to classify the processes as wiping or benign based on NE_P value. In WiDeS, to determine Th, we used Confidence Interval (CI). We assume that the entropies of the processes are normally

distributed; thus, we use CI to define the range of values likely to include the actual population of interest (benign processes). However, if the values deviate significantly from the range, we consider them outliers. We define CI as $CI = \mu_{NE} \pm z\sigma_{NE}$, here μ_{NE} and σ_{NE} are the mean and standard deviation of all NEs associated with benign processes computed by using Equation 3.7 and 3.8 respectively. In WiDeS, we considered CI with 90% acceptance, for which the value of z is equivalent to 1.645). By using μ_{NE}, σ_{NE} and z we now compute CI.

$$\mu_{NE} = \frac{1}{N}\sum_{P=1}^{N} NE_P \tag{3.7}$$

$$\sigma_{NE} = \sqrt{\frac{1}{N-1}\sum_{P=1}^{N}(NE_P - \mu_{NE})^2} \tag{3.8}$$

CI range is defined by $[CI_{lb}, CI_{ub}]$ where CI_{lb} is the lower bound, $CI_{lb} = \mu_{NE} - z\sigma_{NE}$ and CI_{ub} is the upper bound, $CI_{ub} = \mu_{NE} + z\sigma_{NE}$. However, to detect wiping, we consider only CI_{lb} as the entropy values corresponding to wiping processes are far less than benign processes, as shown in Table 3.5. Hence, Th is equivalent to CI_{lb}, i.e., Th=CI_{lb}. If the entropy NE_P is less than Th, i.e., $NE_P < Th$, we detect the process as a wiping process. Conversely, the other processes are classified as benign processes. We create the baseline behavior of the system with benign processes. We use the mean μ_{NE} and σ_{NE} of benign processes to compute CI and determine Th.

3.5.2.2 Filtering Driven by Write() System-call

In this module, we examine the processes classified as wiping (from the first module) to determine if any process is falsely classified. As shown in Algorithm 8, it receives the input as Dict{ptn, freq} corresponding to processes classified as wiping. For each of these processes, we fetch the pattern with maximum frequency ($ptn_{maxFreq}$) from the dictionary $Dict\{ptn, freq\}$ associated with each process.

Subsequently, we retrieve the list of system-calls from $ptn_{maxFreq}$, which can be represented as $ptn_{maxFreq} = s_k,\ s_{k+1},\ s_{k+2}$. Since wiping activities primarily involve writing operations, we expect the $ptn_{maxFreq}$ to include a write() system-call. Therefore, we check if s_k or s_{k+1} or s_{k+2} corresponds to a write() system-call. If the $ptn_{maxFreq}$ indeed contains a write() system-call, it suggests that the process will likely belong to the wiping category. Conversely, the process is classified as benign if no write() system-call is present in $ptn_{maxFreq}$.

Algorithm 8 Fetch pattern with maximum frequency

```
procedure GetPtnWithMaxFreq(Dict{ptn,freq})
    maxFreq ← 0
    ptn_maxFreq ← NULL
    for all ptn, freq in Dict do
        if maxFreq < freq then
            maxFreq ← freq
            ptn_maxFreq ← ptn
    return ptn_maxFreq
```

Table 3.6 lists the processes determined as wiping in the first module. The table shows that processes host, dig, userdel, and vi contain high-frequency patterns but do not contain the write() system-call. On the other hand, processes Shred, SRM, Scrub, Wipe, and Sfill contain write() system-call. Thus, we classify Shred, SRM, Scrub, Wipe, and Sfill as wiping processes and other processes, vi, host, and dig, as benign.

Table 3.6: Processes classified as wiping (from module 1) and their patterns with maximum frequencies

Process	Pattern with Max Frequency	Frequency
Host	mmap mmap mmap	61
Dig	mmap mmap mmap	61
userdel	stat stat stat	109
vi	select select select	110227
Shred	write write write	3584
SRM	write write write	15428
Wipe	write write write	16623
scrub	write write write	1666
sfill	write write write	1002928

3.5.2.3 Analysis of Buffer Data Entropy

In the second module, there is a possibility that a benign process may include a write() system-call in $ptn_{maxFreq}$. In such cases, a benign process is misclassified

as a wiping process. To address this, we aim to distinguish between the benign and wiping processes based on the write() system-call arguments. The write() system-call is invoked by user applications to use kernel services to write content to a file. The syntax for write() system-call [80] is given as,

size_t write(int fd, const void buf[.count], size_t count);

The write() system-call contains three parameters, i.e., file descriptor, buffer, and count. The file descriptor represents the open file to which the content will be written. The buffer contains the data to be written to the file; it temporarily holds the content shared by I/O devices or other processes. The count represents the no.of bytes of data to be written from the buffer to the file.

In WiDeS, we consider the second parameter, the buffer data, to detect wiping. It is an array of characters where each character is equivalent to 1 byte, i.e., 8 bits. We observed that the character array contains Octal Escape Sequences (OES) in wiping processes. An OES contains a backslash followed by one, two, or three octal digits (0-7) like \377 (see Examples 3, 4, 5 for reference).

We initially check if OESs are present in the buffer data to detect wiping. If the probability of occurrence of OES $p(OES)$ is greater than or equal to the probability of occurrence of ASCII characters $p(ASCII)$, i.e., $p(OES) >= p(ASCII)$, we suspect wiping activity. However, to confirm wiping, we proceed further to check if buff[] contains a specific character, as demonstrated in Example 3; or if it contains a sequence of random characters, as illustrated in Example 4; or if it contains repeated patterns of characters, as exemplified in Example 5. Once either condition is met, the process is classified as wiping.

Example 3. *This example shows buff[] data for write() system-call with a specific character. Here, '377' is an octal notation, and '\' signifies the start of an escape sequence. Each octal value '377' represents a byte in octal notation equivalent to 255 in decimal and '11111111' in binary; this ensures that all bits are set to 1 in a byte. The below write() system-call signifies wiping, where all the bytes in the data are set to 1s in binary notation.*

```
write(4,"\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377
\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377
\377\377"..., 28672) = 28672
```

Below is another example of wiping with a specific character; here, the string "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU" consists of the character 'U' repeated 16,384 times, i.e., the entire content is written with the character 'U'.

```
write(3, "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU"..., 16384) = 16384
```

Example 4. *This example of write() system-call shows wiping with random characters. Here, the buffer data contains both ASCII characters and octal values. Each backslash followed by digits represents an octal value. Here,\206, \3, \21, etc., are octal values. Other characters like 'y', '9', 'V', 'c', 'u', 'a', '%', '4', 'A', 'b', 'v', 'J', 'h', 'c' are ASCII characters. We can see that all the characters in the buffer data are random, signifying the wiping process.*

```
write(3, "\277\332\231\335y9\347\273V\206\3\21\225\16\226c\342u
\300\331a%4Ab\306\322\200vJhc"..., 16384) = 16384
```

Example 5. *This example of write() system-call shows wiping with a repeated pattern of characters. These characters include octal values \266 and \333, and ASCII character 'm'. The sequence of '\266\333m' is repeated in the buffer data multiple times, signifying the wiping process.*

```
write(3, "\266\333m\266\333m\266\333m\266\333m\266\333m\266\333m
\266\333m\266\333m\266\333m\266\333m\266\333"..., 28672) = 28672
```

3.5.3 WiDeS Workflow

As mentioned earlier, WiDeS contains three modules. The first module classifies the process as benign or wiping based on the entropy values. The second module classifies the processes identified as wiping in the first module and conducts further classification by determining the presence of write() system-call in a pattern with maximum frequency. In module 3, we further exploit the arguments (buff[]) corresponding to write() system-call to ascertain if the process belongs to wiping or benign. Figure 3.16 gives a brief of WiDeS workflow.

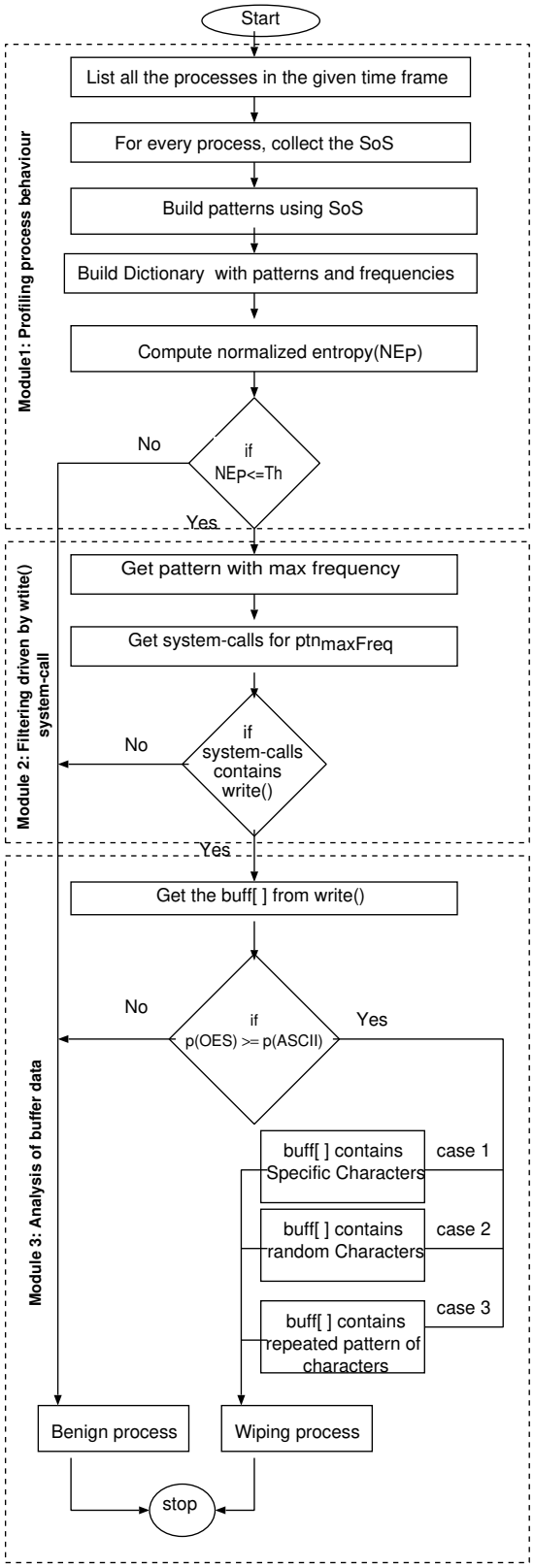


Figure 3.16: WiDeS Workflow

3.5.4 Results and Discussion

We assess WiDeS by applying it to our repository of processes. To construct this repository, we execute 79 benign processes 5 wiping processes, and gather their respective SoS using the Sysdig tool. Additionally, we collected SoS for wiping processes Shred, Scrub, and Wipe by varying the number of passes as 3, 5, 7, and 35 following data wiping standards. For SRM tool, we collected SoS in two modes, i.e., regular and fast modes. By altering the number of passes for each wiping process, we collected SoS for 15 processes. We denote Shred with 3-passes as Shred-3, and Shred with 35 passes as Shred-35; similarly, Scrub-35 signifies wiping with Srcub using 35 passes.

To determine the accuracy of WiDeS, we compute the total number of correct observations (i.e., True Positive (TP), True Negative (TN)) with the total no. of observations (i.e., TP, TN, False Positives (FP), and False Negatives (FN)). We compute the accuracy using Equation 3.9.

$$Accuracy(\%) = \frac{TP+TN}{TP+FP+TN+FN} * 100 \tag{3.9}$$

In the context of wiping, we define TP, TN, FP, and FN as follows: (1) When a process is wiping, and WiDeS identifies it as wiping, it is TP. (2) If a process is benign and WiDeS identifies it as benign, it is TN. (3) If the process is benign, but WiDeS identifies it as wiping, it is FP. (4) If the process is wiping, but WiDeS identifies it as benign, it is FN. FP causes inconvenience to the users by misinterpreting a benign process as a wiping process. On the other hand, FN is more dangerous as the wiping process is detected as benign.

Using WiDeS, we could determine three characteristic behaviors corresponding to the wiping process; they are- (1) the wiping process has less entropy, (2) the pattern with maximum frequency will include write() system-call, (3) the buff[] of the write() system-call will contain random, or specific, or repeated pattern of characters. Based on these characters, we constructed a decision tree for WiDeS. Figure 3.17 shows the decision tree for WiDeS. We map three decision-making conditions to three modules of WiDeS.

The initial decision is taken based on the entropy values. We now compute the accuracy of WiDeS for classification based on entropies by applying WiDeS to our repository of processes. If the entropy $NE_P <= Th$, we determine the process as wiping (see Section 3.5.2.1 for details); results based on this condition show that WiDeS identified 19 processes as wiping. Of these 19 processes, 15 belong to wiping (i.e., TP = 15). Additionally, 4 benign processes (vi, userdel, host, dig)

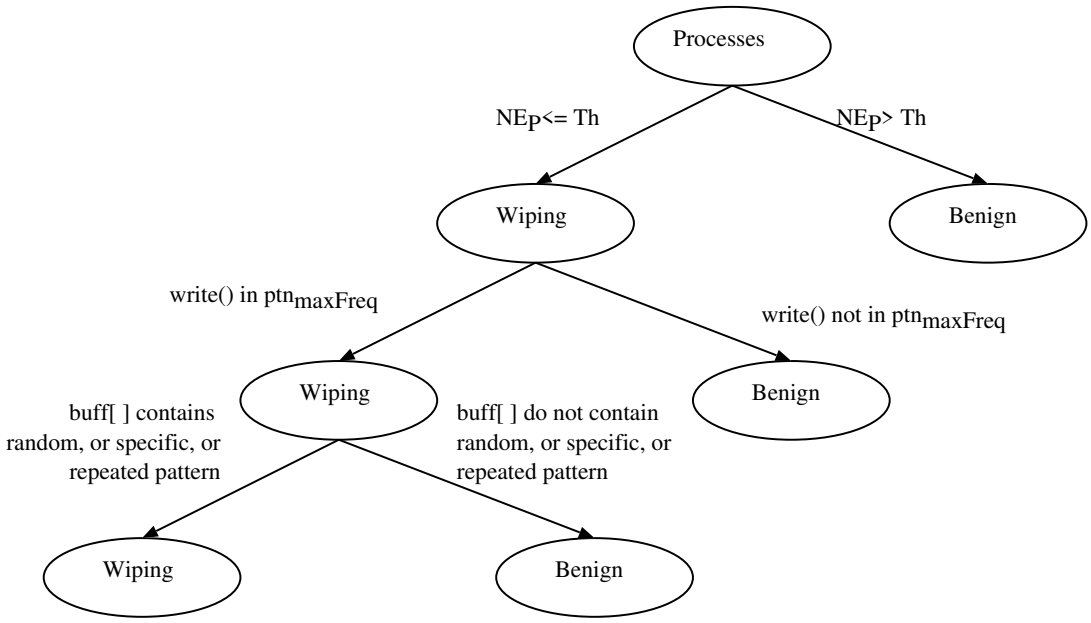


Figure 3.17: Descision tree for WiDeS

were incorrectly classified as wiping (i.e., FP = 4). Other 75 benign processes were correctly identified as benign (i.e., TN = 75), and no wiping processes were mistakenly classified as benign (i.e., FN = 0). Upon computing the accuracy using Equation 3.9 for these values, we get an accuracy of 92.78%.

Figure 3.18 shows the distribution of entropy values across the processes; the data points plotted below the threshold line represent wiping, and the data points plotted above the threshold line represent benign processes. From Figure 3.18, we can see four data points corresponding to benign processes misinterpreted as wiping. The processes vi, userdel, host, and dig are detected as wiping processes due to the presence of some patterns with high probability. Table 3.7 shows patterns and frequency of processes falsely classified as wiping; from this table, we can infer that benign processes with repetitive patterns are classified as wiping processes falsely. We further classify the wiping process using the second decision based on the write() system-call to avoid false positives.

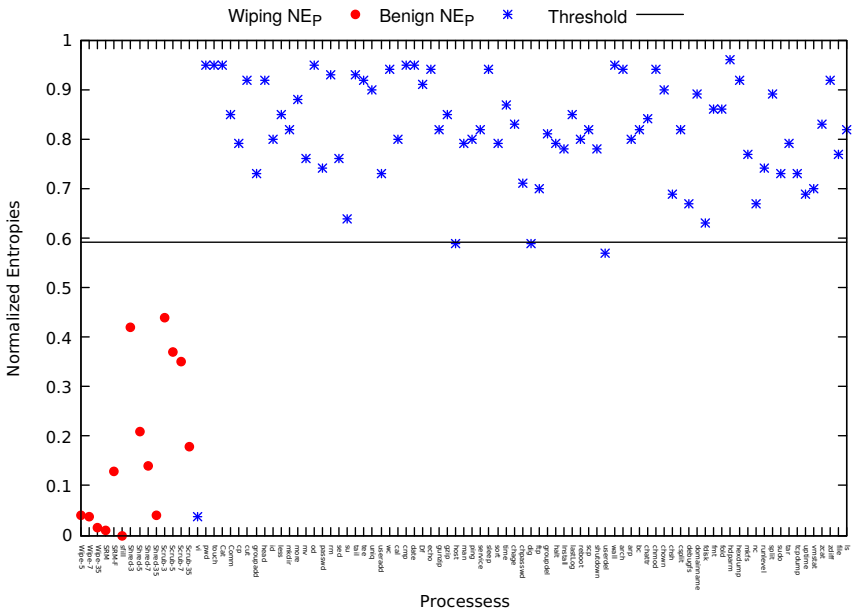


Figure 3.18: Benign and wiping processes entropies

The second decision is taken based on the presence of write() system-call in the pattern with maximum frequency ($ptn_{maxFreq}$). If $ptn_{maxFreq}$ contains write() system-call, we classify the process as wiping (see Section 3.5.2.2 for details). At this stage, we get 19 processes identified as wiping based on the first decision. For these 19 processes, we check if the $ptn_{maxFreq}$ contains the write() system-call. It is observed that the four processes vi, userdel, host, and dig do not contain write() system-call (see Table 3.6 for reference). These four processes were initially classified as wiping based on the first decision. However, they are now identified as benign processes using the second decision. The remaining 15 processes identified as wiping in the second module are further examined using the subsequent third decision based on buffer data.

The third decision is based on the buffer data content (i.e., write() parameter). If the write() parameter buff[] contains random characters or a specific character or a repeated pattern of characters, we determine the process as wiping (see Section 3.5.2.3 for details). Here, we get 15 processes identified as wiping based on the second decision. We observe the buff[] data of the write() system-call to determine wiping. We observed that all the 15 processes classified as wiping contained the buffer data as shown in Example 3, 4, and 5; thus, all the 15 processes were correctly identified as wiping processes.

Table 3.7: List of benign processes, falsely classified as wiping processes with high-frequency patterns

Process	Pattern	Frequency of Patterrn
Host	mmap close openat	31
	close openat read	31
	openat read fstat	29
	read fstat mmap	29
	fstat mmap mmap	22
	mmap mmap mmap	61
	mmap mmap close	31
	mprotect mprotect mprotect	29
Dig	mmap close openat	31
	close openat read	31
	openat read fstat	29
	read fstat mmap	29
	fstat mmap mmap	22
	mprotect mprotect mprotect	29
	mmap mmap mmap	61
	mmap mmap close	31
userdel	stat stat stat	109
	mmap mmap mmap	23
vi	select select write	1218
	select write select	1226
	write select read	1039
	read select select	1325
	select select read	298
	select read select	1335
	select select select	110227
	write select select	187

At the first decision, we had 94 processes as input for classification; at the second decision, we had 19 processes for classification; and at the third decision, we had 15 processes for classification. At every decision, we reduce the input size and get output that is more relevant to wiping. This improves the accuracy after every stage; also, as the input size is reduced, it leads to faster processing.

From the results, we notice that after the first decision, we get an accuracy of 92.78%; WiDeS classified 75 processes as benign and 19 as wiping, of which 4 benign processes were wrongly identified as wiping. However, by the end of the third decision, WiDeS classified 79 processes as benign and 15 as wiping processes with 100% accuracy. Thus, WiDeS determined wiping with 100% accuracy for the given 94 processes. We believe that WiDeS can classify wiping with acceptable accuracy in real-time scenarios.

3.6 Summary

In this chapter, we propose two different models, i.e., WiDeJ and WiDeS. In WiDeJ (static approach), we exploited the file system journaling and the data blocks corresponding to the file to detect file wiping while investigating the cloud VM (post the occurrence of the incident). In WiDeS, we used system-calls to detect wiping while the attack is active by capturing the system-calls of the VM using logging agents like sysdig.

In WiDeJ, we analyze the journal of cloud VM snapshot and fetch the data blocks corresponding to a file. Further, based on the file content, we categorized file wiping as case 1- wiping a file with a specific character, and case 2- wiping a file with random characters. We used Shanon's entropy and file system layout to detect wiping. We computed the file entropy using the frequency distribution of ASCII characters; further, to reduce false positives, we used file layout to analyze the file header to detect wiping. We evaluated the proposed model WiDeJ using 54 files by varying file type. Initially, we could determine wiping with 90% accuracy. Further, based on file layout, we reduced the false positives and could determine wiping with 100% accuracy.

In WiDeS, we used system-calls captured from cloud VMs to determine file wiping. Our proposed model, WiDeS, presents a comprehensive and effective approach to classify processes as benign or wiping using system-calls and information theory metrics. By utilizing three distinct modules, WiDeS achieves a multi-step classification process that enhances the accuracy and reliability of its results. The

first module lays the foundation by initially classifying processes as benign or wiping using entropy values as a primary criterion. This module broadly categorizes processes and is a starting point for further analysis. The second module takes the processes identified as wiping in the first module and delves deeper into their behavior. It conducts additional classification by analyzing the presence of the write() system-call in the pattern with maximum frequency. This approach helps to differentiate wiping processes more precisely from benign ones, adding an extra layer of refinement to the classification process. Lastly, the third module extends the analysis even further by examining the arguments (buff[]) corresponding to the write() system-call. By exploiting these arguments, WiDeS gains valuable insights into the process's nature and additional evidence to determine whether it belongs to the wiping category or is genuinely benign. The three modules work in synergy, enabling WiDeS to achieve accurate classification of wiping processes from benign ones.

Thus, in this chapter, we proposed two different models to detect file wiping. In both the models, we were able to detect wiping accurately. However, WiDeJ had few limitations such as its dependency on journals and file header. We could avoid such dependencies in WiDeS. Further, in the upcoming chapters, we discuss how to recover the wiped files and preserve them to ensure the integrity of the evidence.

Chapter 4

Recovery of Wiped Files

In Chapter 3, we discussed how to detect file wiping in the cloud VMs using static and dynamic approaches. In this chapter, we discuss our next objective, i.e., recovery of wiped files from cloud VMs. We propose deploying our scripts in a cloud VM that runs in the background to monitor and log cloud VM activity for the purpose of recovering wiped files. We exploit the data recovery mechanisms of file systems to restore the wiped files. Data recovery mechanisms in file systems are used to restore them to a consistent state in case of power failure, system crashes, or hardware errors. Major data recovery mechanisms used in file systems include file system journaling and CoW. Thus, we explored a journal-based file system (Ext3) and CoW-based file system (BTRFS) for restoring wiped files.

As discussed in the earlier chapter, a journal in the file system keeps track of changes not yet committed to the disk. By recording such changes, the file system recovers quickly in case of a system crash. Updates in the file system are captured in the journal as transactions. Each transaction contains details corresponding to the file's metadata including data block addresses allocated to a file. We traverse back through these journal entries to fetch the data block address corresponding to a file and restore deleted or previous versions of a file. However, data recovery is possible when the journal is analyzed at the appropriate time; this is due to the cyclic queue data structure of the journal, and the journal content gets overwritten. Ext(3/4), NTFS, ResierFS, and XFS (Extended File System) are a few journal-based file systems.

On the other hand, BTRFS is a CoW-based file system. In a CoW-based file system, when a file is updated or modified, instead of overwriting the existing file, the file's content is copied to a new location, and the changes are

done at the new location. Thus, the original file remains unchanged. This allows multiple file versions to be stored simultaneously, which can be useful for recovering file system to a consistent state incase of an unexpected system crash. Examples of a few CoW-based file systems are (Write Anywhere File Layout), ZFS (Zettabyte file System), and BTRFS. We used the BTRFS file system, an open-source CoW-based file system for Linux. It includes checksums, metadata duplication, snapshots, and RAID support built into the file system.

We emphasize file recovery from BTRFS more, considering the increased adoption of BTRFS for storage environments. BTRFS is widely being adapted and is considered as the next major file system for Linux distributions [81], [82], [83]. Currently, it is the default file system for Fedora [84] and OpenSUSE [85]; it is given as an option for many Linux distributions. Facebook [86] deploys it for millions of servers to increase resource utilization and efficiency. The NAS (Network Attached Storage) storage servers by Netgear [87], Rockstor [88], Synology [89] are using BTRFS for fault-tolerance, easy maintenance, and data protection. Docker [90] and Canonical [91] are using it for container management. Considering increased usage of BTRFS file system, we exploited it for the benefit of recovering wiped files.

4.1 Challenges in Recovering Wiped Files

Journal Based Recovery

In the context of forensics, journal entries can be used to identify recent activities of the file system. Analyzing a file system journal at the appropriate time can help recover previous file versions and deleted files [37]. Gregorio, in his paper [73] details the behavior of the Ext2 and Ext3 file system upon deleting a file. Ext2 does not support file system journaling, unlike Ext (3/4). Thus, he details the procedure to recover deleted files using both Ext2 and Ext3 file systems. In the Ext2 file system, the metadata corresponding to files is used for data recovery, whereas the file system journal is used in the Ext3 file system. Kevin, in his paper [92] provides a comprehensive analysis of the Ext4 file system for data recovery from a forensic perspective. He details the behavior of various data structures upon file deletion. He proposed using Ext4 journal to gather empirical evidence and recover deleted data from persistent data captured from file system changes.

Existing literature has demonstrated the recovery of deleted files from the journal by utilizing previous file versions. However, there has been a lack of emphasis on recovering wiped files using journals. Here, we demonstrate the recovery of wiped files from the journal and discuss its limitations in detail, thus emphasizing the need to look for a CoW-based file system for the purpose of recovering wiped files.

BTRFS Forensics

BTRFS addresses the challenges of the latest storage solutions. However, in the realm of forensics BTRFS has not gained the necessary attention. The existing literature for BTRFS has a handful of papers that provide in-depth details about file system layout and underlying data structures-[17], [82], [93], [94], and [95]. As per our literature, there are only a few papers in the context of BTRFS forensics, which include [81], [83], [96].

Bhat et al., in their paper [81], detail BTRFS artifacts that are rich in information for forensics. They detail the scope of recovering files considering B-trees node balancing in the file system. Hilgert et al. in [83] emphasizes the shortfall of the existing tools for forensics in BTRFS and adds support to the BTRFS file system to TSK tool. It also extends support for multiple device configurations. Only one paper on BTRFS is available in the context of anti-forensics [96]; it explains BTRFS's capabilities to stand against anti-forensic approaches such as file wiping and data hiding using checksum and CoW features of BTRFS file system. BTRFS checksums prevent data, metadata corruption, and data hiding. Similarly, BTRFS withstand wiping using the CoW feature. Multiple versions of the same file exist as the original content remains untouched.

BTRFS is a modern file system with limited support from established forensic tools. Although it is used in well-established organizations and is available as a default/optional file system in Linux distributions, it falls short of tools for proper forensic analysis. Considering the increased utilization of the BTRFS and the limited literature in the context of anti-forensics in BTRFS, we propose to fill this gap by addressing the issues corresponding to file wiping in BTRFS. This work's primary findings contribute in the direction of recovering the wiped and deleted files from the BTRFS file system in the context of anti-forensic practices.

4.2 Contributions

We name our proposed model as *ReWinD* i.e., ***R**ecovering **W**iped and **D**eleted files*. In this direction, the significant contributions of this work include,

- An approach to restore wiped files using BTRFS utility program btrfs-progs.
- A novel approach to restore the wiped files in BTRFS by logging the physical address of the file using superblock.
- Comparing the scope of data recovery between the journal-based file system and the BTRFS file system.
- Usecase of ReWinD for recovery of the unencrypted version of the files following a ransomware attack.

4.3 Prelimnaries

4.3.1 BTRFS Chunks

BTRFS uses extents for efficient file management, unlike in early Linux-based file systems where the disk space was divided into blocks. To avoid file fragmentation and improve the performance of the file system, extents are used in the latest file systems. Extent is a collection of contiguous blocks. This enables the storage of large files without any file fragmentation. These extents are logically separated as BTRFS chunks. Each chunk may contain one or many extents. Based on the type of data stored in them, these chunks are classified as SYSTEM-CHUNK, METADATA-CHUNK, and DATA-CHUNK.

- The SYSTEM-CHUNK is used by the BTRFS file system for the initial Physical Address (PA) to Logical Address (LA) mapping when the system is bootstrapped.
- METADATA-CHUNK holds the metadata; it includes information such as inodes, timestamps, extent offsets, backup information, device information, checksum, etc. In the BTRFS file system, all the trees used to traverse the file system are considered as metadata and are stored in METADATA-CHUNK.
- DATA-CHUNK includes the user files stored on the disk using the file system.

4.3.2 BTRFS Trees

- Chunk tree: The chunk tree contains the logical start address and size of the chunks in the file system. It contains details corresponding to the SYSTEM, METADATA, and DATA chunks. This tree is used to perform logical address-to-physical address mapping.
- Root tree: The root tree contains the logical root addresses of all other trees in the file system.
- File System (FS) tree: FS trees store information about the user files and their metadata, such as inode, timestamps, offset values, etc.
- Extent tree: The extent tree holds the information corresponding to the extent allocation to the files.
- Device tree: The device tree performs PA to LA mapping. Also, it holds the details corresponding to the different devices configured on the BTRFS file system.
- Checksum tree: The checksum tree stores the checksum values and ensures the integrity of the file system.

Tree	Object ID of Tree
ROOT_TREE	1
EXTENT_TREE	2
CHUNK_TREE	3
DEV_TREE	4
FS_TREE	5
CSUM_TREE	7

Table 4.1: BTRFS trees and their object IDs

4.3.3 BTRFS Data Structures

BTRFS file system contains three crucial data structures [97]: 1) BTRFS header, 2)BTRFS key, and 3) BTRFS item . The nodes in the tree are classified as internal nodes and leaf nodes. Each internal node contains BTRFS header, followed by BTRFS key, and block pointer (to point to the next subsequent node) [98]. The

Leaf node contains BTRFS header, followed by BTRFS item, and BTRFS data corresponding to each item. Figure 4.1 shows the leaf node layout in BTRFS file system.

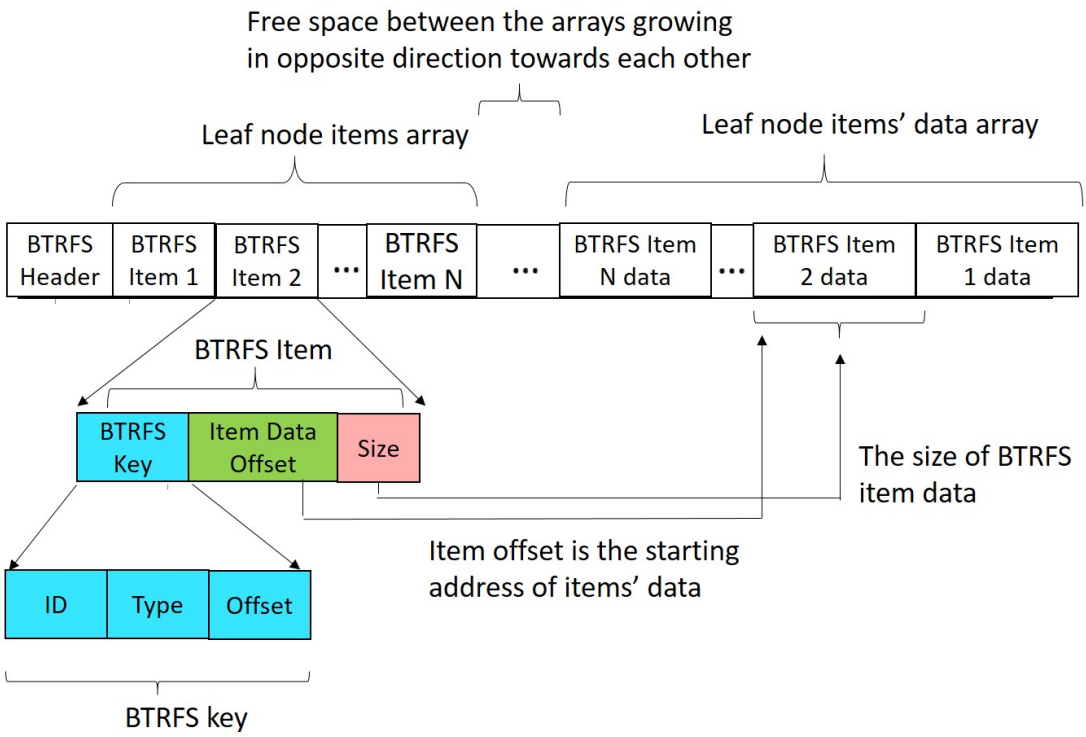


Figure 4.1: BTRFS key and BTRFS item in leaf node

BTRFS header contains fields like checksum, FS UUID, tree LA, tree ID, generation, level etc. We can differentiate between the leaf nodes and internal nodes by the value stored in the block headers 'level' field. If the level field contains the value 1, it signifies the internal node; else, if it contains 0, it signifies the leaf node. Figure 4.2 shows the BTRFS header data structure layout.

The BTRFS items are stored in the leaf nodes; each item has a specific significance. The associated data with each item varies based on the type of item. The BTRFS key is an integral part of the BTRFS item; it contains the object ID, the type of the item, and the offset, as shown in Figure 4.1. The offset value in the BTRFS key depends on the item type; for example, in the chunk_item, the offset value in the key signifies the LA of the chunk. We have listed a few important items and their hex values in Table 4.2; we determine the item type

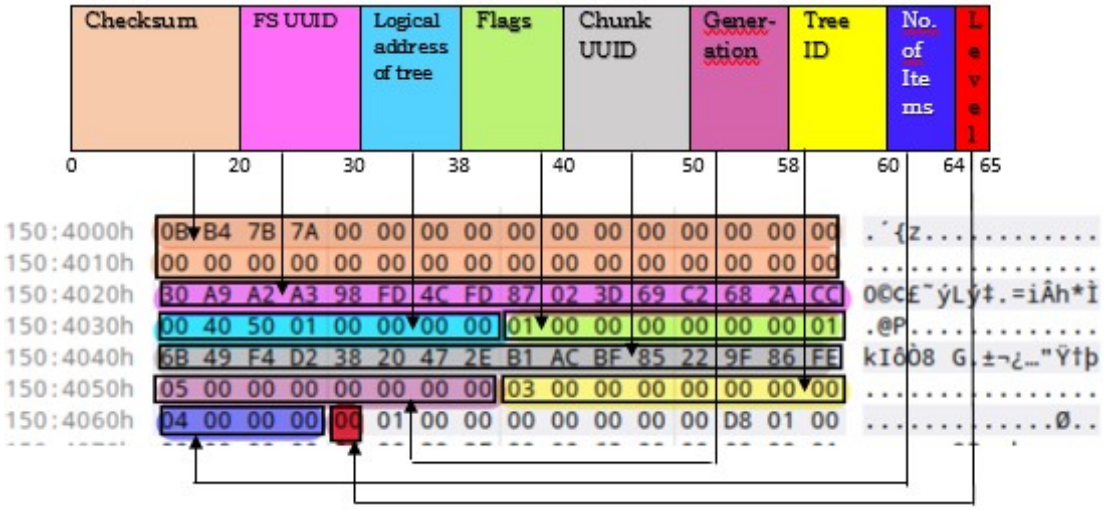


Figure 4.2: BTRFS header of chunk tree

based on these values. For example, let us consider the chunk tree leaf node as shown in Figure 4.3; the item type in this example is E4, which signifies it is a chunk_item (see Table 4.2).

Item type	Hex Value
ROOT_ITEM	84
DIR_ITEM	54
DIR_INDEX	60
INODE_ITEM	01
CHUNK_ITEM	E4
DEV_ITEM	D8
EXTENT_DATA	6C

Table 4.2: BTRFS Tree items

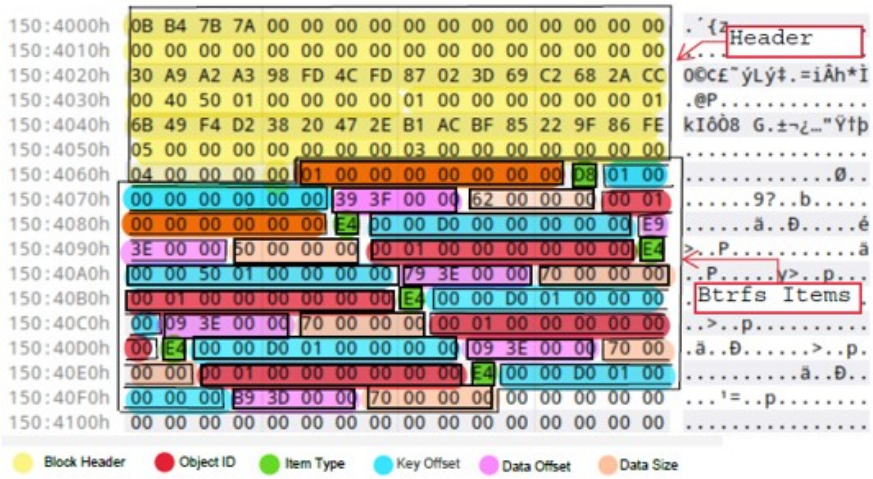


Figure 4.3: BTRFS items captured for a chunk tree leaf node

BTRFS Traversal

BTRFS traversal begins at the superblock, similar to other Linux file systems. The superblock is considered the core file system component that holds the metadata corresponding to file systems. It helps in file system traversal by giving high-level overview of how the data is organized on-disk. The following are the steps involved in BTRFS traversal. Further, in section 4.5.2 we shall discuss file system traversal in detail.

- Step 1: Locate the Superblock at its physical address.
- Step 2: Capture the LA of the chunk tree and root tree from the superblock.
- Step 3: Capture the FS tree LA from the root tree.
- Step 4: Capture the LA of the chunk items for METADATA-CHUNK from the chunk tree.
- Step 5: Compute the PA of the FS tree.
- Step 6: Locate Extent Data item in File system tree.
- Step 7: Compute the PA of the data block.

4.4 Recovery Using Journals

We used Ext3 journal in data journaling mode (see Chapter 3 for details on journaling modes) as it captures the metadata and data blocks allocated to the file. Figure 4.4 shows journal entries in data journaling mode. To analyze the activities performed on the file system, we need to exploit every transaction captured in the journal. To parse the journal, we used jls command-line tool from The Sleuth Kit (TSK); further, to read the data block content allocated to a file, we used blkcat command-line tool from TSK.

Figure 4.4: Traversing Ext3 journal entries to recover previous versions of the file

Now let us try to recover a wiped file using the journal. For this, we first create a file demo.txt with some content; then, we modify the file content, and lastly, we wipe the file demo.txt using the wiping tool Shred. We now try to analyse all the journal and fetch the relevant transactions (i.e., file creation, file modification, and file wiping) to recover the wiped file. Figure 4.4 highlights the transactions for the above actions. The seq: 11 in the journal entries, as shown in Figure 4.4, represents the transaction corresponding to the creation of file demo.txt. The highlighted block 9216 corresponds to a data block allocated to file demo.txt. This block will store the file content. We read the block contents at

9216 using the blkcat tool. We now fetch the journal transaction corresponding to file modification, i.e., seq: 29 from the journal. We can see that the file demo.txt now has block 9729 allocated to it. We use the blkcat command-line tool to view the contents at block 9729. Figure 4.4 shows the content of the file after modification at block 9729. Further, Seq: 44 from the journal transaction represents the activity corresponding to file wiping. Upon viewing the block content 9729 after wiping, we see that the file content is completely overwritten with '0's. However, we can still recover the previous version of the file from block 9216, which was captured by the journal earlier during demo.txt file creation.

We were able to recover wiped files from the journal, but the journal has a cyclic queue data structure with a fixed size; the journal entries get overwritten as the file system utilization increases. Thus, recovery using a journal is only feasible till the journal entries corresponding to the files are available. Thus, we explored the CoW-based file system, BTRFS, for data recovery.

4.5 Proposed Model ReWinD

In ReWinD , we propose two different approaches to recover wiped and deleted files. In the first approach, we use an existing BTRFS utility tool, btrfs-progs, and in the second approach, we propose to log the PA of the files upon their creation by traversing the BTRFS file system. We explain these approaches in detail in the subsequent sections.

4.5.1 ReWinD Using btrfs-progs

In ReWinD we use btrfs-progs utility to recover deleted and wiped files. Btrfs-progs is a utility program for the BTRFS file system. It contains a set of command line tools used to manage and display internal structures of BTRFS. This approach is beneficial as we leverage the default tools of BTRFS for the purpose of forensics without depending on third-party forensic tools. We parse data structures of BTRFS using btrfs-progs utility and traverse the file system to locate the targeted file's data blocks.

To demonstrate recovery of the wiped file, we created an image of 1.1 GB with BTRFS file system and btrfs-progs utility on it. Subsequently, we created a file F1.txt with some content, and later wiped it using Shred tool. Let us now see the various steps involved in ReWinD for file recovery using btrfs-progs utility.

1. The first step is to fetch the contents of the superblock using the btrfs-progs utility command, i.e., dump-super, as shown below. This lists the contents of the superblock in an organized manner. We divided the output generated in two different Figures i.e., Figure 4.5 and Figure 4.6 for improved readability. We further discuss superblock data structure in detail in Section 4.5.2, Figure 4.10 can be referred for the generic overview of superblock data structure. A few important fields from superblock considered for ReWinD include root, chunk_root, generation (see Figure 4.5), and backup_roots (see Figure 4.6). Here, root and chunk_root display the LA of the root tree and chunk tree LA.

```
$ btrfs inspect-internal dump-super -f image.dd
```

```
└─$ btrfs inspect-internal dump-super -f dfr-10-ext.dd
superblock: bytenr=65536, device=dfr-10-ext.dd
---------------------------------------------------------
csum_type               0 (crc32c)
csum_size               4
csum                    0x82ebef13 [match]
bytenr                  65536
flags                   0x1
                        ( WRITTEN )
magic                   _BHRfS_M [match]
fsid                    7bca6d3c-0713-40b2-ac38-8dc3bec21e39
metadata_uuid           00000000-0000-0000-0000-000000000000
label
generation              12
root                    31145984
sys_array_size          129
chunk_root_generation   5
root_level              0
chunk_root              22036480
chunk_root_level        0
log_root                0
log_root_transid (deprecated)   0
log_root_level          0
total_bytes             1073725440
bytes_used              135168
sectorsize              4096
nodesize                16384
leafsize (deprecated)   16384
stripesize              4096
root_dir                6
```

Figure 4.5: Superblock output using btrfs-progs

2. We now analyse backup_roots of superblock. The superblock contains backup_roots to ensure file system consistency. The four backup_roots are labeled as backup 0, backup 1, backup 2, and backup 3 (see Figure 4.6). Each backup contains the BTRFS tree root node LAs, generation number, and level. The generation numbers are used to track the changes. They get incremented for any change in the corresponding tree, the new LAs of root nodes are captured, and the corresponding generation number is incremented in the backups. Superblock stores the most recent four generations of backup_tree_root(i.e., LA of root tree). For any change in the file system, if the root tree LA gets updated, the backup with the minimum generation number in the superblock gets overwritten with the latest LA of the trees.

```
backup_roots[4]:
        backup 0:
                backup_tree_root:       30867456        gen: 10 level: 0
                backup_chunk_root:      22036480        gen: 5  level: 0
                backup_extent_root:     30851072        gen: 10 level: 0
                backup_fs_root:         30769152        gen: 10 level: 0
                backup_dev_root:        30556160        gen: 6  level: 0
                csum_root:      30785536        gen: 10 level: 0
                backup_total_bytes:     1073725440
                backup_bytes_used:      131072
                backup_num_devices:     1

        backup 1:
                backup_tree_root:       31129600        gen: 11 level: 0
                backup_chunk_root:      22036480        gen: 5  level: 0
                backup_extent_root:     31113216        gen: 11 level: 0
                backup_fs_root:         30883840        gen: 11 level: 0
                backup_dev_root:        30556160        gen: 6  level: 0
                csum_root:      30998528        gen: 11 level: 0
                backup_total_bytes:     1073725440
                backup_bytes_used:      135168
                backup_num_devices:     1

        backup 2:
                backup_tree_root:       31145984        gen: 12 level: 0
                backup_chunk_root:      22036480        gen: 5  level: 0
                backup_extent_root:     31162368        gen: 12 level: 0
                backup_fs_root:         30883840        gen: 11 level: 0
                backup_dev_root:        30556160        gen: 6  level: 0
                csum_root:      30998528        gen: 11 level: 0
                backup_total_bytes:     1073725440
                backup_bytes_used:      135168
                backup_num_devices:     1

        backup 3:
                backup_tree_root:       30752768        gen: 9  level: 0
                backup_chunk_root:      22036480        gen: 5  level: 0
                backup_extent_root:     30736384        gen: 9  level: 0
                backup_fs_root:         30720000        gen: 9  level: 0
                backup_dev_root:        30556160        gen: 6  level: 0
```

Figure 4.6: Root backup from superblock

3. Now, we try to fetch the LA of the FS tree from the backup root nodes. Select the backup whose backup_fs_root generation number is maximum; it contains the LA of the FS tree for the current state of the file system. For example, from Figure 4.6, we can see that backup 2 contains the generation number gen: 11, which is maximum compared to other backup_fs_root generation numbers; but backup 1 also contains backup_fs_root with gen: 11. This is because there were no changes in the FS tree but the other trees may have been modified; in this example the extent tree has been updated. Thus, the extent tree generation number is incremented to 12, which also increments the root tree generation number to 12. In scenarios where the FS tree generation number remains the same across multiple backups, as shown in Figure 4.6, we can fetch the backup_fs_root from either backup 1 or 2, which is equivalent to 30883840.

4. Once we have the LA of the FS tree, we try to capture the data blocks corresponding to the targetted file by analyzing the FS tree. For this, we parse the FS tree using btrfs-progs utility command dump-tree at LA 30883840 as shown below. Figure 4.7 shows the items of the FS tree. The dir_index lists the available files and sub-directories. Further, we use inode_item and extent_data of the file names listed by dir_index items to compute the PA of the data blocks corresponding to the file. Here, we compute the PA of file F1.txt. The computation of PA using the LA is discussed in detail in the section 4.5.2.

```
$ btrfs inspect-internal dump-tree -b 30883840 image.dd
```

```
└─$ btrfs inspect-internal dump-tree -b 30883840 dfr-10-ext.dd
btrfs-progs v6.6.3
leaf 30883840 items 7 free space 15635 generation 11 owner FS_TREE
leaf 30883840 flags 0×1(WRITTEN) backref revision 1
fs uuid 7bca6d3c-0713-40b2-ac38-8dc3bec21e39
chunk uuid 7cffb064-cdf6-408b-a092-4176ad98f0fb
        item 0 key (256 INODE_ITEM 0) itemoff 16123 itemsize 160
                generation 3 transid 10 size 12 nbytes 16384
                block group 0 mode 40755 links 1 uid 0 gid 0 rdev 0
                sequence 21 flags 0×0(none)
                atime 1703932590.876732441 (2023-12-30 05:36:30)
                ctime 1703932589.176231862 (2023-12-30 05:36:29)
                mtime 1703932589.176231862 (2023-12-30 05:36:29)
                otime 1703932335.0 (2023-12-30 05:32:15)
        item 1 key (256 INODE_REF 256) itemoff 16111 itemsize 12
                index 0 namelen 2 name: ..
        item 2 key (256 DIR_ITEM 892701812) itemoff 16075 itemsize 36
                location key (265 INODE_ITEM 0) type FILE
                transid 10 data_len 0 name_len 6
                name: F1.txt
        item 3 key (256 DIR_INDEX 11) itemoff 16039 itemsize 36
                location key (265 INODE_ITEM 0) type FILE
                transid 10 data_len 0 name_len 6
                name: F1.txt
        item 4 key (265 INODE_ITEM 0) itemoff 15879 itemsize 160
                generation 10 transid 11 size 4096 nbytes 4096
                block group 0 mode 100644 links 1 uid 0 gid 0 rdev 0
                sequence 10 flags 0×0(none)
                atime 1703932702.626144117 (2023-12-30 05:38:22)
                ctime 1703932684.572766039 (2023-12-30 05:38:04)
                mtime 1703932684.572766039 (2023-12-30 05:38:04)
        item 5 key (265 INODE_REF 256) itemoff 15863 itemsize 16
                index 11 namelen 6 name: F1.txt
        item 6 key (265 EXTENT_DATA 0) itemoff 15810 itemsize 53
                generation 11 type 1 (regular)
                extent data disk byte 13639680 nr 4096
                extent data offset 0 nr 4096 ram 4096
                extent compression 0 (none)
```

Figure 4.7: FS tree dump using btrfs-progs

5. As we have the PA of the data blocks corresponding to a file, we now analyze the hex dump of the image at the computed PA. The Figure 4.8 shows the contents of the file F1.txt. This shows that the file content is wiped.

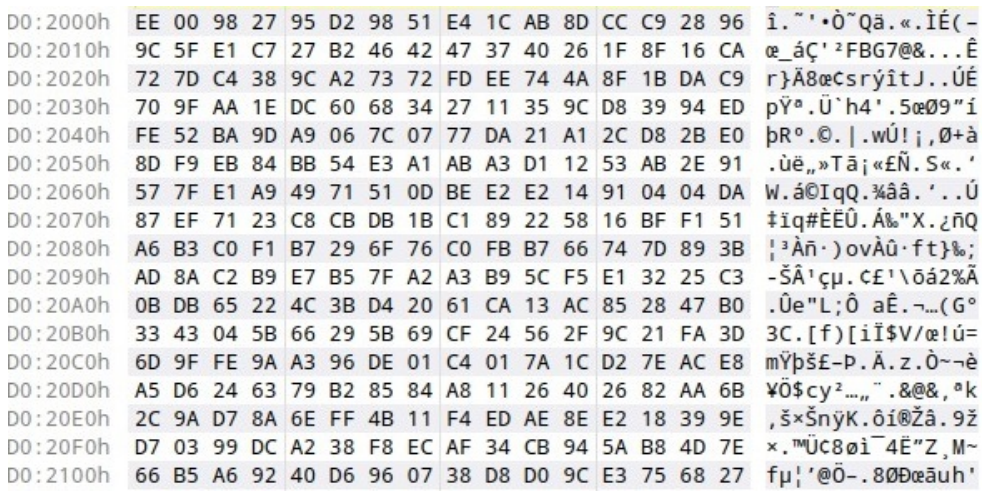


Figure 4.8: File data after wiping

6. Now, we try to fetch the previous version of the file to recover the original contents of file F1.txt. For this, we look at the superblock backup_roots for a backup_fs_root whose generation number is one less than the current backup_fs_root generation number i.e., 11. This is because the earlier generation number would contain the metadata corresponding to the previous version of the file. For example, in backup 0, the backup_fs_root with gen: 10 has the LA of backup_fs_root equivalent to 30769152, which refers to the earlier state of the file system, which may contain the previous version of the file F1.txt. The limitation of this approach is if the generation number of the current backup_fs_root is the same in all four backups, then we cannot recover the previous version of the file as the reference to the previous version is lost. This happens when there are modifications in other trees of the file system with no changes in the FS tree.

7. Now we again parse the FS tree at LA 30769152 and locate the PA of the data blocks of the targetted file F1.txt. This gives us the PA of the data blocks corresponding to the previous version of the file. We parse the FS tree for dir_index, inode_item , extent_data for file F1.txt. Further, we compute the PA for file F1.txt and locate the file content on the Disk. Figure 4.9 shows the content of the file before wiping (i.e., for backup_fs_root: 30769152 and gen:10), and Figure 4.8 shows the content of the file after wiping (i.e., for backup_fs_root: 30883840 and gen:11). Thus, we can recover wiped files by traversing through the previous state of the file system.

```
255:BE10h  00 00 00 00 00 00 00 00 00 00 0A 00 00 00 00 00  ................
255:BE20h  00 00 2D 00 00 00 00 00 00 00 00 00 00 00 00 48  ..-............H
255:BE30h  65 6C 6C 6F 21 21 20 54 68 69 73 20 69 73 20 66  ello!! This is f
255:BE40h  69 6C 65 20 46 31 20 61 66 74 65 72 20 6D 6F 64  ile F1 after mod
255:BE50h  69 66 69 63 61 74 69 6F 6E 2E 20 0A 0B 00 00 00  ification. .....
255:BE60h  00 00 00 00 06 00 46 31 2E 74 78 74 0A 00 00 00  ......F1.txt....
255:BE70h  00 00 00 00 0A 00 00 00 00 00 00 00 2D 00 00 00  ............-...
```

Figure 4.9: File Content before wiping

Thus, we recovered the wiped files using btrfs-progs. However, it depends on the availability of FS tree LA from backup_roots of the superblock. If we have frequent changes in the file system, then we lose reference to the previous state of the file system as the backups get overwritten, and the file recovery becomes difficult. To overcome this limitation, we propose our next approach, i.e., ReWinD, by logging PA of the files.

4.5.2 ReWinD by Logging PA of Files

In the BTRFS file system, like many other Linux-based file systems, traversal begins at the superblock. Superblock is a predefined data structure with a fixed location on the disk. Even though the BTRFS file system is a CoW-based file system, the superblock is not CoWed, as frequent changes in the PA of the superblock may result in an inconsistent file system. Thus, the superblock content is overwritten for changes in the file system. We can access all the files in the file system at a given time T using the superblock's contents. Let us understand this better by considering an example.

Example 6. *Data blocks corresponding to file F_1 at time T_1, can be retrieved by computing the PA of the file using the contents of the superblock SB_1. Now let us assume that at time T_2, the contents of the file F_1 have been modified; being a CoW-based file system, it creates a copy of file F_1 as F_1' and updates the changes to F_1'. As there are changes in the file system, the superblock is updated to SB_2. Using the contents of SB_2, one can retrieve the PA of file F_1' but cannot retrieve PA of the original file F_1. If we want to fetch the contents of the file F_1, i.e., the previous version of the file, then we need to have SB_1. If the contents of SB_1 are available, one can still compute the PA of the file F_1, but the contents of the superblock SB_1 at time T_1 is overwritten as SB_2 at Time T_2; thus, the reference for the file F_1 is lost.*

From the above Example 6, we understand the possibility of recovering the previous version of the file provided we have the PA of the file. Therefore, it is proposed to compute the PA of the files upon their creation using superblock contents and log the PA of the file along with the file's metadata for future reference. We check the superblock content(i.e., root tree LA) for every 30 seconds because the superblock undergoes updates at this frequency [17]. If there are no alterations in the file system, the contents of the superblock remain unchanged. Thus, for every 30 seconds, we check for the root tree LA in the superblock. If the root tree LA is updated, we traverse the file system and log the PA of newly created files and their corresponding metadata, else we wait for the next 30 seconds and check for updates. Thus, by recording the PA of the files, we can traverse the file system back in time to recover deleted and wiped files.

Thus, we write a script to compute and log the PA of the files and deploy them on the cloud VMs. These scripts run in the background and monitor the file system activity of cloud VMs. Upon the creation of files, the script computes the file's PA and logs the file's PA and its metadata. Later, in the future, we can refer to the log file for the corresponding file PA and recover the wiped file content.

A. BTRFS Traversal Using Superblock

As we compute the PA of a file using the superblock, let us understand how we traverse the file system using the superblock without using any utility programs and third-party tools. We initially analyze the superblock data structure located at a pre-defined location on disk, i.e., 0x10000 [94], and parse its contents. Figure 4.10 shows the superblock data structure layout. Further, we fetch the following from the superblock to compute PA- 1) sys_chunk_array, 2) Chunk Tree LA, and 3) Root tree LA. Let us now understand the significance of each item listed in detail below,

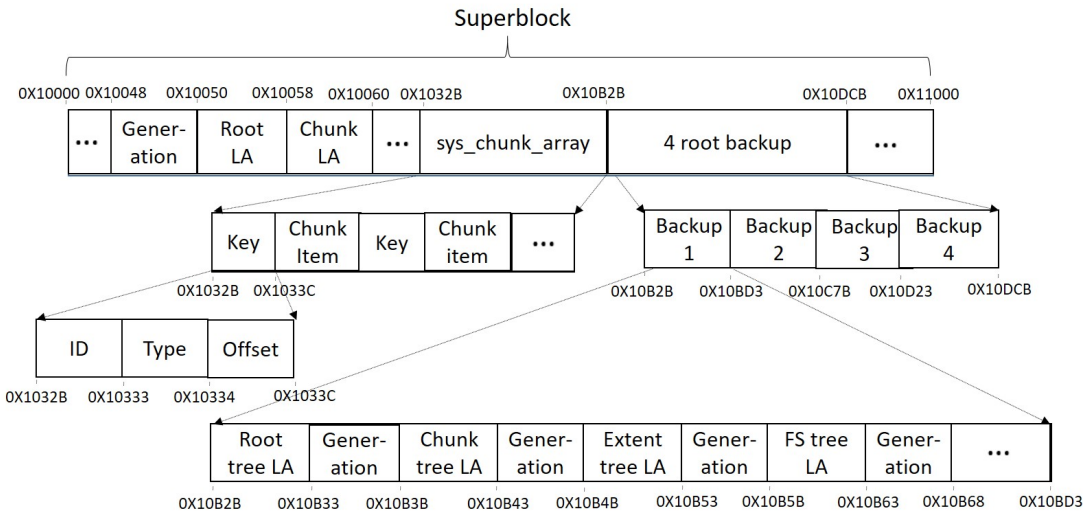


Figure 4.10: Superblock layout for bootstrapping the file system

1) The chunk tree is responsible for LA to PA mapping. The chunk tree resides in the SYSTEM-CHUNK. To get the PA of the chunk tree, we use chunk_item from sys_chunk_array of superblock. Thus, we initially parse the sys_chunk_array in the superblock and compute the PA of the chunk tree.

2) The chunk tree contains the chunk_items corresponding to SYSTEM-CHUNK, METADATA-CHUNK, and DATA-CHUNK. These chunk_items contain the logical start, size, and other metadata corresponding to each chunk. All the trees in BTRFS reside in the METADATA-CHUNK. We use the chunk_item corresponding to METADATA-CHUNK to compute the PA of other trees using chunk tree.

3) The root tree contains the LA of root nodes associated with different trees in BTRFS file system. We get the LA of the FS tree from the root tree. Further, using the LA of the FS tree, we capture the user files and directories.

B. Compute Physical Address

To compute the PA of a tree (PA_T), we need the following values- 1) LA of the tree (LA_T), 2) the logical start of the chunk (LA_C), and 3) the stripe offset. A stripe is associated with a chunk item, representing a portion of allocated space on the disk. Each stripe contains information such as its starting offset (where it begins on the disk). We get LA_T from root tree, LA_C, and stripe offset from the chunk tree's chunk_item. To get PA_T we first compute the difference between

LA_T and LA_C, as shown in the Equation 4.1. Since all the trees are located within a logical chunk (i.e., METADATA-CHUNK), we calculate the difference between the chunk's starting address and the tree's address within the chunk, i.e., Δ. Later, we add Δ to the stripe offset as shown in Equation 4.2 to get PA_T.

$$\Delta = LA_T - LA_C \tag{4.1}$$

$$PA_T = \Delta + StripeOffset \tag{4.2}$$

After computing the PA_T we look for BTRFS items and their corresponding offset data in the leaf node. Based on the type of the item, the offset data differs. To compute the PA of BTRFS item's offset data, we use Equation 4.3. Here, PA_D is the PA of the data, size(header) is the size of BTRFS header, which is equivalent to 101 bytes. The data_offset corresponds to the BTRFS item's data in the leaf node at the given offset (see Figure 4.1 for reference).

$$PA_D = PA_T + size(header) + data\_offset \tag{4.3}$$

C. Compute the Physical Address of the Chunk Tree

To compute the PA of the chunk tree using Equation 4.1 and 4.3 we need LA_T of the chunk tree, LA_C of SYSTEM-CHUNK, and its corresponding stripe offset. Let us further discuss the details specific to each of them and the sources to fetch these values in detail.

We get the LA_T of the chunk tree from the superblock at PA 0x10058 address (see Figure 4.10). Additionally, Figure 4.11 shows the hex dump of the superblock with root tree LA and chunk tree LA. From Figure 4.11 we observe that the LA of the chunk tree LA_T is 0x00040501000000 (in big-endian); upon converting it in little-endian it is equivalent to 0x0000000001504000, and it's decimal equivalent is 22036480 i.e., $LA_T = 22036480$.

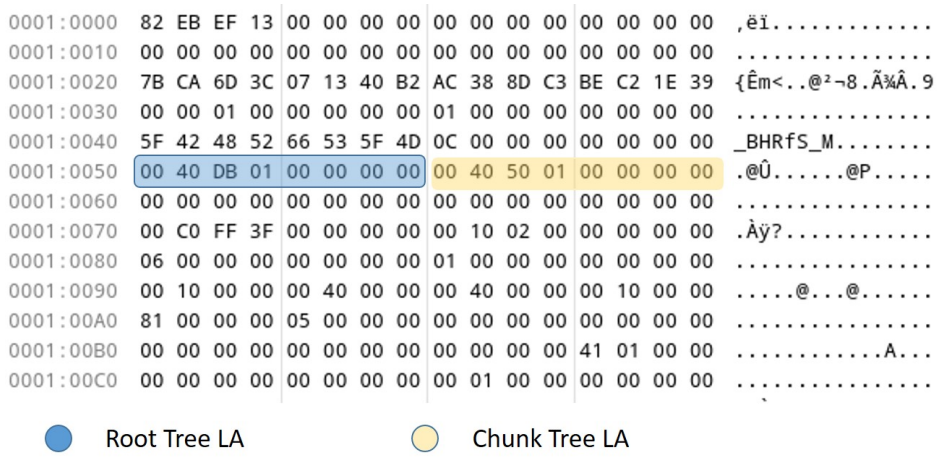


Figure 4.11: Root tree and chunk tree LA from superblock

Further, to compute the LA_C, we need to fetch the details corresponding to SYSTEM-CHUNK because the chunk tree resides in SYSTEM-CHUNK whereas all the other trees, like root tree, FS tree, extent tree, checksum tree, etc., reside in METADATA-CHUNK. Thus, we need LA_C of SYSTEM-CHUNK and its corresponding stripe offset to compute the PA of the chunk tree. We get LA_C of SYSTEM-CHUNK from sys_array_chunk of superblock. Figure 4.10 shows that the sys_array_chunk begins at physical address 0x1032B. The sys_array_chunk contains the data as a pair of (Key,chunk_items). As discussed earlier, the key contains object id, item type, and offset; here, the type of item is chunk_item, and the corresponding offset value is the LA_C of SYSTEM-CHUNK (see BTRFS items in section 4.3.3 for details). Also, using the sys_array_chunk, we get the stripe offset value. The figure shows the hex dump for sys_array_chunk. The values in the Figure 4.12 are highlighted based on the data structure for chunk_item [94].

Figure 4.12: sys_arr_chunk from superblock

From superblock, we get LA_T of chunk tree (see Figure 4.11), and we get LA_C and stripe offset of SYSTEM-CHUNK from sys_array_chunk (see Figure 4.12). Here, both LA_C and stripe offset values are equivalent i.e., $0x0000500100000000$ (in big-endian) equivalent to 0x0000000001500000 (in little-endian), and its decimal equivalent is 22020096 i.e., $LA_C = 22020096$ and $StripeOffset = 22020096$ now substitute LA_T and LA_C in Equation 4.1 i.e.,

$\Delta = LA_T$ - LA_C
$\Delta = 22036480 - 22020096$
$\Delta = 16384$

Now substitute Δ in Equation 4.2 to get the PA_T of the chunk tree. We observe the PA_T of the chunk tree is equivalent to its LA_T, i.e., 22036480 (i.e., 0x1504000). However, this differs with other trees as the LA_C and physical stripe address differ.

$PA_T = \Delta + StripeOffset$
$PA_T = 16384 + 22020096$
$PA_T = 22036480$

We parse the chunk tree at PA 0x1504000 on disk to get the details corresponding to the METADATA-CHUNK, as it contains other trees of the BTRFS file system. To differentiate between METADATA, DATA, and SYSTEM chunk items in the chunk tree, we look at chunk_item's offset data for the type field as shown in Figure 4.13. For this, we first need to compute PA_D of every chunk item to determine its type. However, here we present computing PA_D for METADATA-CHU NK, similar approach can be adopted to determine the PA_D of other chunk_items(i.e., SYSTEM, DATA chunks). Fetch the data_offset for chunk_item as shown in Figure 4.13 i.e., 0x093E0000(in big-endian); equivalent to 0x00003E09 (in little-endian) and, its decimal equivalent is 15881 i.e., $data\_offset = 15881$. In Equation 4.3, we now substitute $PA_T = 22036480$, $size(header) = 101$, $data\_offset = 15881$.

$PA_D = PA_T + size(header) + data\_offset$
$PA_D = 22036480 + 101 + 15881$
$PA_D = 22052462$

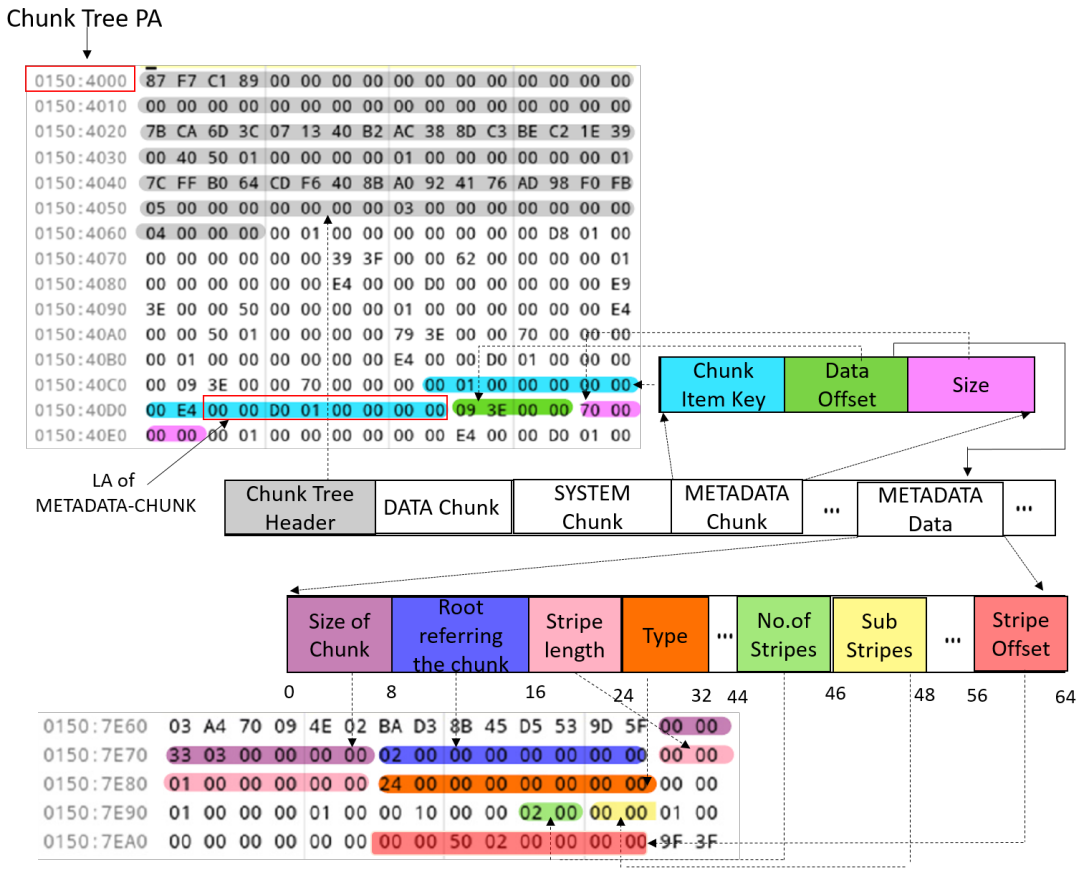


Figure 4.13: Chunk tree leaf node layout

Thus, we get $PA_D = 22052462$ equivalent to 0x1507E6E in hex. From Figure 4.13, we can see that the METADATA chunk_item starts at PA 0x1507E6E. The type field in chunk_item offset contains value 01 for DATA-CHUNK, 02 for SYSTEM-CHUNK, and 04 for METADATA-CHUNK [99]. But, in Figure 4.13, we see the value of the 'type' field for METADATA-CHUNK is 0x24. This is because the type field also includes the information corresponding to data redundancy supported by the file system.

In BTRFS, we associate data redundancy with the following profiles: single, DUP, RAID. In a single profile, the data is stored on a single device without redundancy. In the DUP profile, the data is mirrored (i.e., two copies) of the same data are available. DUP can be implemented on single or multiple devices. In a RAID profile, data redundancy is achieved using RAID features such as stripping, mirroring, and parity; it is suitable when multiple devices are configured. Each

profile has a specific value, e.g., the DUP profile has a value of 32 in decimal and 0x20 in hexadecimal notation [99]. For example, let us say the BTRFS file system uses the DUP profile whose value is equivalent to 20, then the type field of METADATA-CHUNK contains $20 + 04$, i.e., 0x24, which justifies the value shown in the Figure 4.13. The type field for SYSTEM-CHUNK contains $20 + 02$ i.e., 0x22. If we notice that the type value for the DATA-CHUNK is 01. This signifies that the METADATA and the SYSTEM chunks are mirrored, but the DATA-CHUNK is not. Thus, one can configure these settings based on their requirement while initializing the file system.

D. Parsing the Root Tree

After parsing the superblock and the chunk tree, we now parse the root tree. The root tree holds the LA of all other trees' root nodes. The root tree contains the LAs of the file system tree, chunk tree, extent tree, device tree, checksum tree, and others for various file system-relevant activities. We focus on fetching the FS tree LA from the root tree.

To parse the root tree, we initially need to compute the physical address of the root tree. To compute the physical address of the root tree, we need LA_T of the root tree, LA_C of the METADATA-CHUNK, and its corresponding stripe offset. We get the LA_T of the root tree from the superblock (see Figure 4.11). LA_C of the METADATA-CHUNK and stripe offset from the chunk tree (See Figure 4.13).

Figure 4.11 shows the hex dump of root tree logical address from the superblock i.e., 0x0040DB0100000000 (in big-endian); which is equivalent to 0x0000000001DB4000 in little-endian and in decimal is equivalet to 31145984, i.e., $LA_T = 31145984$. From Figure 4.13, we fetch the METADATA-CHUNK logical starting address i.e., offset $LA_C = 0x0000500200000000$ (in big-endian) equivalent to 0x000000025000 (in little-endian) and, its decimal equivalent is 30408704 i.e., $LA_C = 30408704$. Now substitute LA_T and LA_C in Equation 4.1 i.e.,

$\Delta = LA_T$ - LA_C
$\Delta = 31145984 - 30408704$
$\Delta = 7372807$

Now substitute Δ in Equation 4.2 to get the PA_T of the root tree. Here, the stripe offset of METADATA-CHUNK from chunk tree as shown in Figure 4.13 is 0x02500000 (in little-endian), equivalent to 38797312 in decimal.
$PA_T = \Delta + StripeOffset$
$PA_T = 7372807 + 38797312$
$PA_T = 39534592$

Thus, the PA of the root tree $PA_T = 39534592$ is equivalent to 0x25B4000 as shown in Figure 4.14. We now fetch FS tree logical address. For this, we first need to compute PA_D of FS tree in the root tree; thus, we fetch the data_offset for FS tree i.e., 0x653A0000(in big-endian); equivalent to 0x00003A65 (in little-endian) and, its decimal equivalent is 14949 i.e., $data\_offset = 14949$. In Equation 4.3, we now substitute $PA_T = 39534592$, $size(header) = 101$, $data\_offset = 14949$.
$PA_D = PA_T + size(header) + data\_offset$
$PA_D = 39534592 + 101 + 14949$
$PA_D = 39549642$

Thus, from the root tree, we get $PA_D = 39549642$ (equivalent to 0x25B7ACA) of the FS tree; we parse the data at this PA_D to retrieve the LA of the FS tree. From Figure 4.14, we can see that the FS tree LA is 0x0040D70100000000 (in big-endian), equivalent to 0x01D74000 (in little-endian) and 30883840 in decimal. Thus, using the root tree we fetch the LA of the FS tree which is equivalent to 30883840. We now have the LA of the FS tree, but we need the PA of the FS tree to parse the FS tree, for this we again compute the PA of FS tree using the Equations 4.1, and 4.2 as shown below, Here, LA_T of FS tree is 30883840, and $LA_C = 30408704$ (i.e., LA_C of METADATA-CHUNK from chunk tree).
$\Delta = LA_T$ - LA_C
$\Delta = 30883840 - 30408704$
$\Delta = 475136$

$PA_T = \Delta + StripeOffset$
$PA_T = 475136 + 38797312$
$PA_T = 39272448$

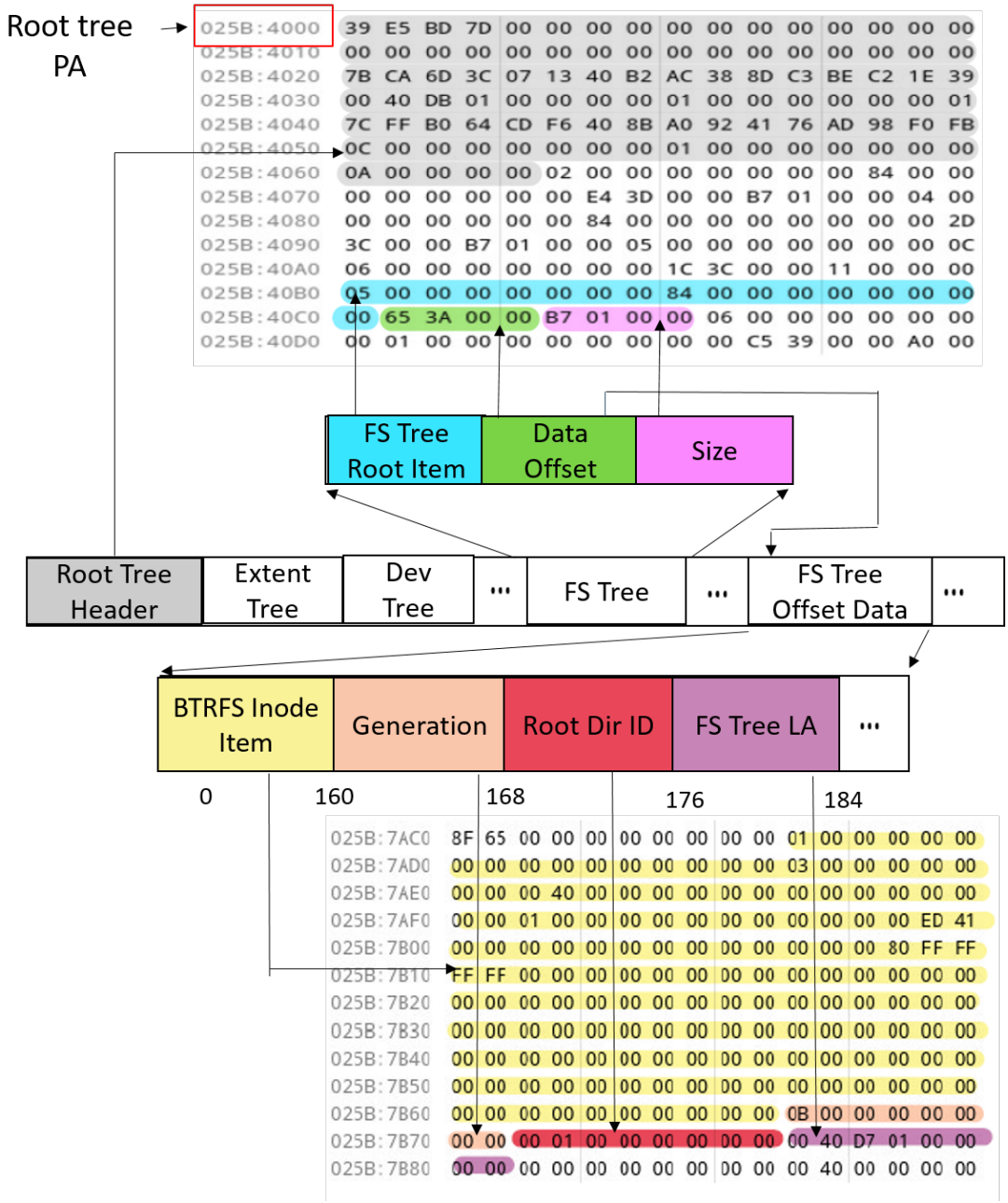


Figure 4.14: Root Tree Leaf Node Layout

Thus, the computed PA of the FS tree is 39272448, equivalent to 0x02574000. Now that we have the PA of the FS tree we can parse the FS tree to get the metadata and data corresponding to files. We further detail the procedure to capture the PA of a file in the subsequent section.

E. Parsing the File System Tree

The File system tree contains the details corresponding to user files and directories. It contains different items corresponding to file data and metadata. e.g., dir_item, dir_index, inode_ref, inode_item, extent_data, etc. Using the FS tree, we try to fetch file metadata and data. Figure 4.15 demonstrates the leaf node layout for FS tree. We use three important items from the FS tree to locate the data blocks corresponding to a file- 1) dir_index, 2) inode_item, and 3) extent_data. From Figure 4.15 we observe that these three items, dir_index, inode_item, and extent_data, have an inode number in common; using the inode number, we correlate these items corresponding to a specific file.

1) The dir_index is used as a lookup for directory entries. The dir_index items' corresponding offset data in leaf node will include the file name, the inode number of the file, and other metadata. We can detect the deleted file entries using dir_index. Let us consider files F1.txt, F2.txt, F3.txt created at time T_1. The FS tree will contain three dir_index entries, each entry corresponding to a file. Now, let us assume that at time T_2, the file F2.txt is wiped or deleted. As there is a change in the file system, these changes are CoWed to a new location. Now, the FS tree dir_index will not contain F2.txt entry. Thus, by comparing the dir_index entries at time T_1 and T_2, we can list the deleted files.

2) The inode_item contains the metadata corresponding to the user file. It contains the inode number, the file size, MAC timestamps associated with the file, and other additional information.

3) The extent_data item contains the LA of the file extent. Along with this, it also includes additional information regarding compression, encryption, generation, transaction, etc., As shown in Figure 4.15. If the file is small, it is accommodated as an inline file in the leaf node at extent_data's offset. For large files, they are stored in the extents; the LA of the file extent is stored in extent_data's offset.

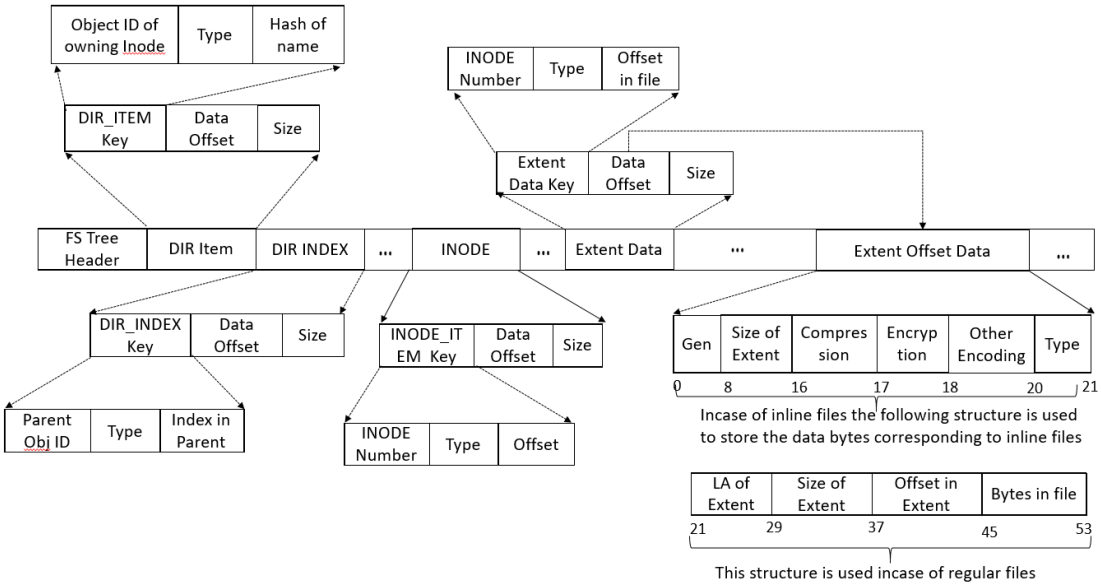


Figure 4.15: Fs tree leaf node layout

To compute the PA of inline files, we consider LA_C of METADATA-CHUNK, but for regular files, we consider LA_C of DATA-CHUNK. This is because the inline files are stored in the leaf node of the FS tree; as discussed in section 4.5.2, the FS tree and other trees of BTRFS are stored in METADATA-CHUNK. on the other hand, regular files are stored in the extents. These extents are stored in the DATA-CHUNK; thus, we use the LA_C of DATA-CHUNK for regular files. We compute the PA of inline files using the Equations 4.3 and for regular files, we use the Equations 4.4, 4.5, and 4.6.

In Equation 4.4, the LA_E is the LA of the extent where the file is stored, and LA_C is the LA of the DATA-CHUNK. We get LA_E from FS tree (see Figure 4.15 for *LA of Extent*) and LA_C from Chunk tree.

$$\Delta = LA_E - LA_C \tag{4.4}$$

We now compute the PA of the extent i.e., PA_E by substituting the Δ from Equation 4.4 in Equation 4.5. We get the stripe offset of DATA-CHUNK from chunk tree.

$$PA_E = \Delta + \text{Stripe Offset of DATA-CHUNK} \tag{4.5}$$

In Equation 4.6, the PA_D contains the PA of regular file on the extent. To compute PA_D we sustitute PA_E from Equation 4.5; and extent_offset from FS

tree (See Figure 4.15 for *offset in extent*) in Equation 4.6 to compute the PA of regular file.

$$PA_D = PA_E + extent\_offset \tag{4.6}$$

In ReWinD, we use the dir_index to list the deleted and wiped files, inode_item to get the files' metadata like inode number and timestamps, and finally, we use the extent_data to retrieve the PA of the files. We have the PA of files and corresponding metadata; we now log them in a separate log file periodically for every 30 seconds, as discussed earlier in the section 4.5.2. Later, we use this log file to recover the PA of wiped or deleted files. However, there is a limitation: the contents of unreferenced files may be overwritten by the file system itself, which is common across all the file systems. Thus, it becomes imperative to determine the scope of recovery using ReWinD which we discuss in the next subsequent section.

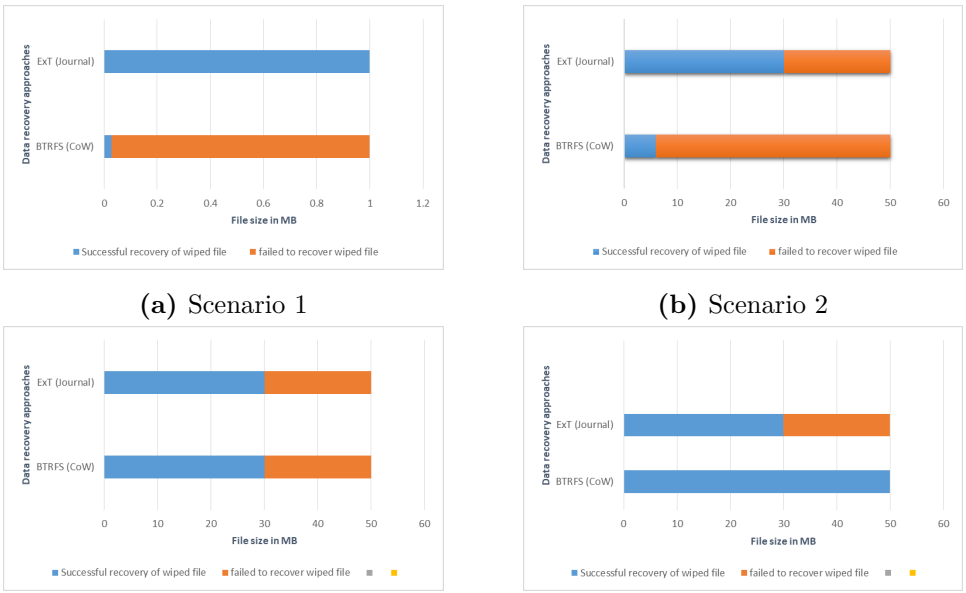
4.6 Results and Discussion

BTRFS increases the scope of recovering wiped files because it is based on CoW principle, but it also comes with the associated challenges. As the entire file system is organized in the form of b-tree, the creation of new file or the deletion of a file may result in node splitting or merging to balance the tree [81]. The merging of nodes especially creates challenges as it overwrites the content when the files are deleted.

To evaluate the proposed model, we have experimented the scope of recovering wiped files in BTRFS by varying the image size, file size, and the DATA-CHUNK size. Table 4.3 details the experimental setup for BTRFS file system for different scenarios. In BTRFS file system, the user files are stored in the DATA-CHUNK. The space allocated to the DATA-CHUNK dynamically increases as its utilization increases.

Scenario	Image Size	File Size	Block group profiles		
			System	Metadata	Data
Scenario 1	1 GB	5 KB to 1 MB	8 MB	51.19 MB	8 MB
Scenario 2	1 GB	5 KB to 50 MB	8 MB	51.19 MB	232 MB
Scenario 3	1 GB	0 KB to 50 MB	8 MB	128.19 MB	532 MB
Scenario 4	3 GB	5KB to 50 MB	8 MB	153.56 MB	2.37 GB

Table 4.3: Experimental setup for different scenarios to evaluate BTRFS



(a) Scenario 1 **(b)** Scenario 2

(c) Scenario 3 **(d)** Scenario 4

Figure 4.16: Comparing the scope of file recovery between Ext and BTRFS under different scenarios as listed in Table 4.3

We further compared the scope of recovering wiped files between Ext (journal-based) file system and BTRFS (CoW-based). For the journal-based file system, we created an image of size 1 GB with the Ext file system, whose journal size is equivalent to 8 MB with default journaling mode, i.e., ordered mode. We used the Shred tool to wipe files using 35 passes.

It is observed that when the allocated DATA-CHUNK unit is small, i.e., 8 MB (scenario 1), it becomes difficult to recover the wiped files. This is because when we wipe a file with 35 passes, for every pass file is copied to a new location and overwritten with specific or random characters. We can say this because

we could observe patterns like '11111', 'UUUUU', and random characters (e.g: 's/ÔïYhpâ@ÁèS') in different locations when analyzing the hex dump of the image. When we wipe the file multiple times there is a possibility that due to the limited space, the original file content is overwritten.

In BTRFS, the unallocated space within the 8MB of the DATA-CHUNK is utilized to overwrite the contents; we say this because upon analyzing the original file's physical address, it was found at 0XD00000, and upon wiping the file with 3 passes we found that the original file content is not overwritten, but the content with specific pattern like '11111111' equivalent to the file size was observed at physical address 0xD01000 followed by specific patterns at location 0xD2000 and 0xD3000. This shows that when a file is wiped, the files system initially uses the unallocated space equivalent to file size. In case of wiping with multiple passes, it may overwrite the file content due to unavailability of space.

Thus the scope of recovery depends upon three factors- 1) image size, 2) DATA-CHUNK size, and file size. Let us assume the size of the DATA-CHUNK is 8 MB, and the file size is 1MB, and if we wipe the file with 35 passes, we cannot recover the wiped file. This is because, during each pass, the 1MB file is replicated to a new location, eventually occupying 35MB of space. but given that only 8MB of space is available, the content gets continuously overwritten, leaving no scope for file recovery. On the other hand, let us consider a file of size 3 KB when wiped with 35 passes, the file of size 3kb is copied to new locations 35 times, consuming space close to 105 KB, but as the size of the DATA-CHUNK is 8 MB we can recover the file. We tested the proposed model by varying the DATA-CHUNK size and the file size. The results for the same are presented in Figure 4.17; with the DATA-CHUNK size equivalent to 232 MB, we were able to recover the wiped files whose file size$< 5MB$. Similarly, for DATA-CHUNK sizes of 532 MB and 1.3 GB, we were able to recover the wiped files whose sizes were less than 30 MB and 50 MB, respectively. This shows that as the DATA-CHUNK size increases, it increases the scope of recovering files with a larger size. The size of DATA-CHUNK depends on the image size. Further, the DATA-CHUNK size increases dynamically based on file system utilization to store user files depending on the availability of unallocated space of the image.

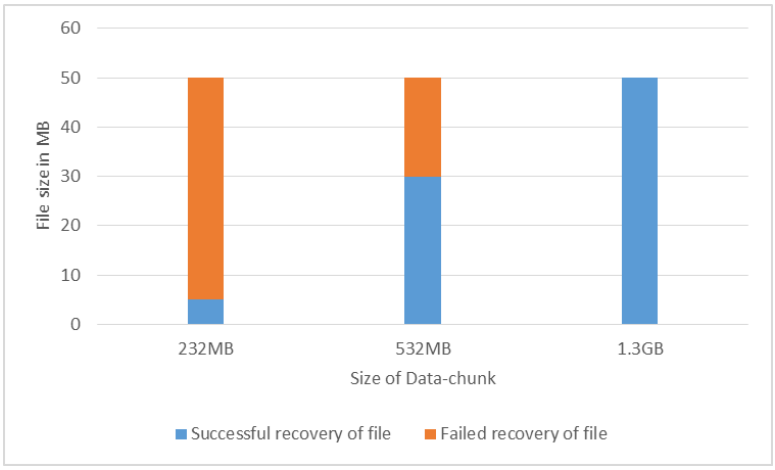


Figure 4.17: Scope of recovering wiped files from BTRFS by varying the DATA-CHUNK and file sizes

Also, we compared the scope of recovering wiped files using the Ext and BTRFS file systems. The results in Figure 4.16 show that the scope of recovery from journals is higher when the file system is newly initialized and eventually decreases. Further, the scope for recovery of the wiped files increases in BTRFS and reduces in Ext as the file system ages; this is because the journal size is fixed when the file system is initialized; but, in BTRFS file system, the DATA-CHUNK size changes based on file system utilization.

4.7 Usecase: Recovery of a file encrypted by Gonnacry ransomware

In this section, we present a usecase by applying the proposed model ReWinD to recover the files affected by ransomware attacks. We try to recover encrypted files without using any decryption keys. To test the application on ReWinD on ransomware we create an image file of 1.1 GB with BTRFS file system. We create a file Hello.txt of size 31 bytes, with DATA_CHUNK size of 8MB. We used Gonnacry, a Linux-based ransomware, to launch the ransomware attack. It initially traverses different paths of the file system and lists the files that can be encrypted. Once a list is created, it encrypts the files sequentially. The files encrypted by Gonnacry have file extension as .GNNCRY, for example the original file Hello.txt is renamed as Hello.txt.GNNCRY.

```
backup 2:
        backup_tree_root:       30507008        gen: 7  level: 0
        backup_chunk_root:      22036480        gen: 6  level: 0
        backup_extent_root:     30457856        gen: 7  level: 0
        backup_fs_root:         30441472        gen: 7  level: 0
        backup_dev_root:        30605312        gen: 7  level: 0
        backup_csum_root:       30490624        gen: 5  level: 0
        backup_total_bytes:     1073725440
        backup_bytes_used:      147456
        backup_num_devices:     1

backup 3:
        backup_tree_root:       30670848        gen: 8  level: 0
        backup_chunk_root:      22036480        gen: 6  level: 0
        backup_extent_root:     30638080        gen: 8  level: 0
        backup_fs_root:         30621696        gen: 8  level: 0
        backup_dev_root:        30605312        gen: 7  level: 0
        backup_csum_root:       30490624        gen: 5  level: 0
        backup_total_bytes:     1073725440
        backup_bytes_used:      147456
        backup_num_devices:     1
```

Figure 4.18: Superblock after encryption

Let us use ReWinD with the btrfs-progs utility (see section 4.5.1 for details) and try to recover the unencrypted version of the file using the backup root nodes of the superblock. Figure 4.18 shows the backup root contents of the superblock. Considering the space constraint, we have not included backup 0 and backup 1 in Figure 4.18. We try to fetch the details corresponding to the current state of the file system. Thus, we capture the details corresponding to the latest generation number of the FS tree root node. Form the Figure 4.18 we can see that backup 3 contains the latest generation number i.e., gen: 8 with $LA_T = 30621696$. Subsequently, we parse the FS tree using its LA_T and list the files using the dir_index item. Here, we find a file with file name as Hello.txt.GNNCRY. We suspect this file to be encrypted; further, we get the inode_item and extent_data item for this file as shown in Figure 4.19b to compute the PA of the file. We compute the PA of the files using the Equation 4.1, 4.2 and 4.3 as discussed in section 4.5.2. Figure 4.20b shows the file contents of Hello.txt.GNNCRY at its PA; we see that the file content is encrypted. Thus, we try to get the previous original version of the file Hello.txt using the superblock backup root nodes whose generation number is one less than the current generation number(gen: 8) i.e., gen: 7.

From Figure 4.18, we see that the FS tree root node LA at generation number gen: 7 is 30441472. Next, we parse the FS tree and lookup for the file Hello.txt as shown in Figure4.19a. We get the inode_item and extent_data item corresponding to the file Hello.txt and compute its PA. Figure 4.20 shows the original content of the unencrypted file. Thus, the CoW principle enables the BTRFS file system

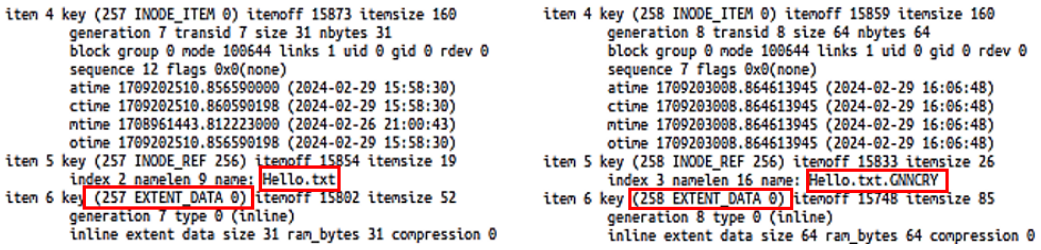


Figure 4.19: FS Tree before and after encrypting the file using Gonnacry

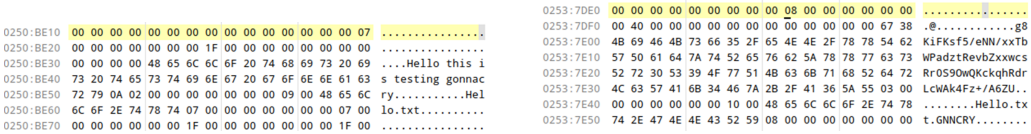


Figure 4.20: File content at physical address before and after encryption

to hold the original file content even before encryption. By traversing to the previous version of the file we can recover the unencrypted file content.

This usecase of ReWinD still needs to be analyzed rigorously under different complex ransomware attack scenarios. In our future work, we shall try to explore the restoration of encrypted files from ransomware attacks under different scenarios.

4.8 Summary

We introduce ReWinD, a novel model for recovering wiped and deleted files in cloud VMs backed by BTRFS file system. We have analyzed two distinct methods for recovering wiped files from the BTRFS file system. Initially, we utilized the existing BTRFS utility program, btrfs-progs, for file recovery. During this process, we observed that the superblock stores up to four backups of the root tree. Leveraging this feature allows us to navigate to previous file versions. However, there is a limitation: these backup roots are overwritten for the changes in the file system, causing a loss of data to access the previous file version.

Subsequently, we propose another novel approach, which is logging the PA of files. This method entails logging the PA of files every time the superblock is updated. We maintain a separate log file that captures the PA of the files,

which mitigates the risk of losing reference to the earlier versions of files. We compared the scope of data recovery between Ext and BTRFS under different scenarios, which reveals that BTRFS has greater potential to recover wiped files as the file system ages. Furthermore, we demonstrate the application of ReWinD on a ransomware attack launched by Gonnacry, where we recover the previous unencrypted version of the file using the proposed model. This highlights the practical relevance of our work in addressing real-world data recovery challenges, especially in the context of anti-forensics (i.e., artifact wiping) and threats like ransomware.

Chapter 5

Investigation Model to Preserve Cloud VMs and Investigation Proceedings on Blockchain

In this chapter, we provide an approach to preserve the evidential artifact (i.e, cloud VM) in a forensic sound manner such that the integrity of the evidence is preserved using blockchain technology. Involving blockchain in the preservation of cloud evidential artifacts addresses some additional challenges associated with cloud forensics. The most critical problem is the evidence's validity and trustworthiness when multiple stakeholders are involved [100]. These stakeholders can always collude among themselves and tamper with the evidence [101] for various reasons. Also, centralized ownership of the evidence is with CSP, who cannot be trusted completely. To decentralize ownership and ensure evidence's integrity, immutability, authenticity, availability, and transparency among the involved stakeholders, we propose a blockchain-based, tamper-proof, and transparent investigation model using permissioned blockchain.

Blockchain technology is a game-changer in digital forensics, especially when multiple stakeholders are involved [100]. Blockchain is a series of append-only, immutable, transparent data structures (i.e., blocks) that store the details of every transaction on the peer-to-peer network [100] [102]. Blockchain has become the most promising technology for achieving integrity, auditability, transparency, security, authenticity, etc. Thus, the usage of blockchain to ensure cooperation and transparent exchange of information among the authentic stakeholders involved in the investigation makes it an optimal technology for forensics.

5.1 Challenges in Existing System

Let us initially understand the existing investigation procedure in case of an incident on cloud VM. Upon receiving an incident report, the LEA assigns an IO for it. The IO then issues legal notice to CSP to immediately restrain the current services offered by CSP to the suspected VM. Later, IO issues another notice to CSP to provide the requested evidential artifacts like target VM snapshot, relevant logs (system, user activity logs), etc., and preserve the evidential artifacts for future investigation and court trials.

Figure 5.1 shows the phases involved in the existing system cloud incident investigation. It mainly involves; 1. Issuing legal notice to CSP to furnish the required details, 2. Evidence Identification, 3. Evidence Collection, 4. Evidence preservation, 5. Submission of evidence by CSP to LEA, 6. Evidence Analysis, and 7. Reporting and presentation of evidence.

Upon receiving notice from LEA, CSPs identify the cloud resources involved in crime from the pool of resources distributed across the globe involved in the incident. After identifying the evidential resources, CSP collects evidential artifacts across the cloud environment. IO will neither have direct access to the physical resources nor be involved in evidence collection due to various privacy constraints in the cloud. IO ultimately has to depend on the CSPs for evidence. CSP shares evidence with LEA through the proper channel. After the evidence is shared with LEA, both CSP and LEA start the analysis of the evidence for further investigation.

Suppose other stakeholders (forensic analyst, prosecutor, defender) are involved in the investigation and need access to evidence. It takes much time as it involves many legal proceedings to ensure proper Chain of Custody (CoC) and ensure the integrity of evidence. Moreover, the investigation procedure is not transparent to the members involved in the incident. There is always a possibility that the organizations, legal entities, and other members involved in the incident may collude and tamper with the evidence for their benefit, resulting in a lack of confidence in the legal system.

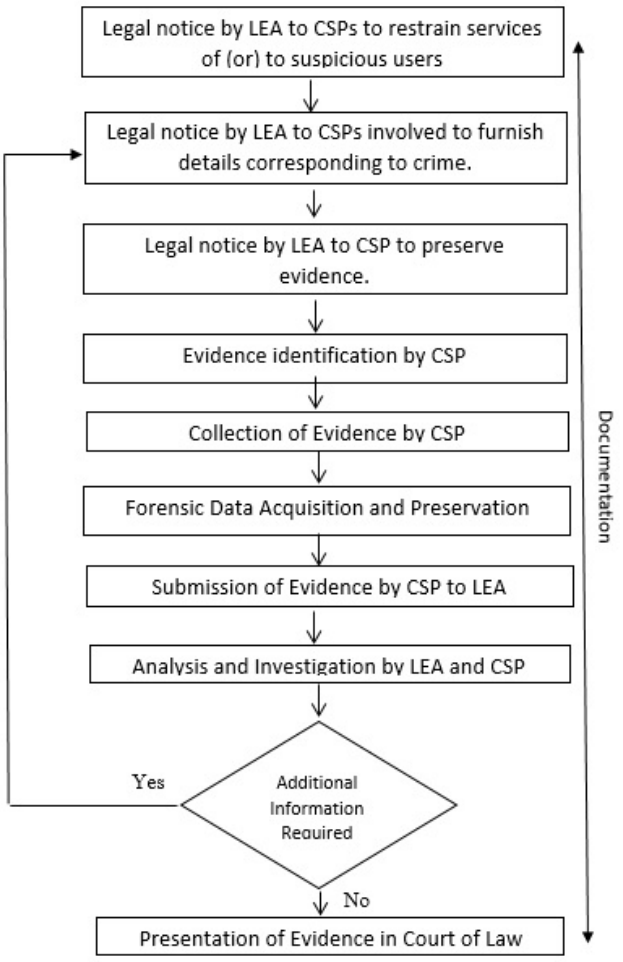


Figure 5.1: Existing Investigation Procedure.

5.2 Contributions

To overcome the existing challenges, as discussed, we propose an investigation model using blockchain. We name our proposed model as Investigation-Chain. The significant contributions of our model Investigation-Chain includes:

- A tamper-proof and transparent investigation model for cloud VMs and ensure the availability of VM snapshots to multiple stakeholders using permissioned blockchain.
- We validated the proposed model using a case study and evaluated its per-

formance for the proof of concept using Hyperledger Caliper.

- Finally, we present a comparative analysis of recent research with our model and prove that our proposed model fulfills all the essential security aspects of evidence preservation compared to other research works.

5.3 Prelimnaries

5.3.1 Blockchain

A decentralized platform for information sharing through a ledger enables multiple authoritative domains that do not trust each other to coordinate, cooperate, and collaborate in decision-making. The information is shared across the network by a ledger. A ledger is an immutable data structure used to capture the transactions chronologically, thereby maintaining the history of all transactions across the network. Based on the access given to the participating nodes in the blockchain network, they are classified as permissionless and permissioned blockchains. The permissionless blockchains are also known as public blockchains; any member providing Proof-of-Work can be included in the network without knowing their identity. Also, the ledger is transparent and open to all network members. On the other hand, permissioned blockchain is also known as a private blockchain, an access control layer that ensures only authorized members access the network.

5.3.2 Hyperledger Fabric (HLF)

HLF [103] is a permissioned blockchain framework to implement private blockchain. HLF contains members who are legally separate entities known as organizations. Each organization can host multiple nodes. Nodes initiate the transaction that leads to the execution of a smart contract. Transaction privacy and confidentiality is ensured by using channels in HLF. A

channel is a subset of peers who want to share the information confidentially.

Components of HLF:

- Certificate Authority: Certificate Authority issues identity and access control certificates using public and private keys to network components and users.

- HLF Client: HLF Client is the client application that interacts with the blockchain. The application can be developed using Software Development Kit (SDK), which supports Node Java Script (JS), Java, Python, and Go languages.
- Chaincode: Chaincode is the smart contract deployed on the blockchain network that details the business logic. The HLF client application invokes and queries the functions on the chaincode. The chaincode is deployed on the peer binaries of the blockchain network.
- Peer: Peers are the blockchain's nodes; they are the fundamental entities of the blockchain that host the ledger and chaincode. An organization can host one or many peers.
  - Anchor Peer: Discoverable outside the organization.
  - Leader Peers: Connect to the orderer to receive new blocks.
  - Endorsing peers: Special peers that sign the transactions before the transaction is committed.
- Orderer: Ensure consistency across the network by determining the sequence of transactions. It also prepares a block of transactions and broadcasts the new block in the network.

HLF workflow: The client application submits a transaction to endorsing peers; the peers will execute the transaction, agree the output is the same across other peers, and add their signature. The client application has to collect endorsements from multiple peers in the network to say it is a valid transaction as all the outputs are the same. Later, submit the transaction for ordering; this ensures the transactions are ordered across all the nodes to validate that no two transactions should try to change the value of a variable simultaneously. If done, then the first transaction is allowed, and the later transaction is invalidated. Orderer broadcasts the new block to peers in the network. The peers validate if the new transaction has received the required endorsements and later commit the transaction. Finally, the committed block is added to the blockchain, and the ledger is updated.

5.4 Application of Blockchain in Digital Forensics

In cloud forensics, evidence accessibility, availability, transparency, and integrity have always been a challenge. Earlier proposed solutions are using blockchain to address these challenges to some extent [104]. Also, most of these works focus on ensuring the integrity and auditability of evidence by storing the hash value and series of timestamps of the forensic artifacts on the blockchain. Most cloud artifacts considered for forensics include cloud logs, CoC, and files' metadata. Let us further discuss the existing solutions for each of these artifacts in detail and summarize the strengths and weaknesses of each work (see Table 5.1).

5.4.1 Log Integrity

Park in his paper [105] proposed a solution for a blockchain-based data logging and integrity management system for cloud forensics. The proposed solution involves blockchain across multiple CSPs. It compares the proposed method with other cryptocurrencies-based blockchains. The log data is collected from instances and encrypted from each CSP. Later, the hash values for the data corresponding to each CSP are generated and stored on the block. The participating CSPs determine the time taken to generate a block. After the permitted time, a block integrity check is performed on each block and added to the blockchain; this ensures data integrity even before the data is used as evidence by investigators using a permissioned blockchain.

The author in paper [101] proposes secure logging as a service using blockchain to avoid altering logs by un-trusted cloud stakeholders. Node Controller (controls the log activity of all virtual machines) collects the logs from all virtual environments and encrypts them. After every day, the Node controller publishes the encrypted logs, the hash of encrypted logs, and the hash of the previous encrypted log on the blockchain; this ensures the integrity, availability, and chronology of the logs. In case of an incident on the cloud, forensic investigators can access the blockchain, decrypt the logs, and access them as mentioned in Service Level Agreement (SLA). The proposed solution achieves integrity, confidentiality, and availability using permissioned blockchain. However, vast volumes of cloud logs are stored on the blockchain, increasing the computational cost and communication overhead.

5.4.2 Metadata Integrity

Liang proposes Provchain [18], which mainly focuses on data provenance of files stored on cloud storage using blockchain. The corresponding file operations, such as file creation, modification, and deletion, are maintained on the blockchain by storing a file's metadata every time an event is performed. Provchain is built on an open-source application ownCloud, to collect provenance data. ownCloud is a web-based cloud storage service. As the user acts on the data files on their ownCloud, the metadata, including username, filename, and actions performed, are recorded. A provenance auditor uses the chainpoint protocol to publish data records on the blockchain network and generate blockchain receipts. The blockchain receipt contains the transaction information and is used to validate the data. Cloud customers have to pay a fee as a reward for blockchain miners to opt for data provenance services.

Tian, in his paper [106], proposes an approach to store the information corresponding to the evidence on the blockchain and store the actual evidence on a trusted storage platform; this avoids blockchain bloat and optimizes the performance of the system.

5.4.3 Chain of Custody

It is found that most of the work in forensics using blockchain focuses on auditability by ensuring the integrity of CoC as proposed in papers [19], [102], [107], [108], [109].

Zhang, in his paper [107], proposes process provenance for the CoC, i.e., proof of existence and privacy of the process records using blockchain and cryptography group signature. The receiver sends a request to the sender to collect forensic data; the sender responds to the request with a submission list. The submission list records the hash values of transferred forensic data files and the group signatures of the sender and receiver. The receiver verifies the hash values of the files and confirms the submission list by signing it. Later, the process auditor verifies the process records using a group signature and collects the records to a certain number. Finally, the process records are published on the blockchain network, and the corresponding blockchain receipt is stored.

The author Bonomi proposes a model B-CoC, i.e., blockchain-based CoC [108] using a permissioned blockchain, where only authorized entities can access the evidence. Only one authorized owner can access or own the evidence at a given

time. If any other authorized entity needs access, the current owner needs to raise a transfer request. This transfer request is recorded on the blockchain as an evidence log. The actual evidence is stored in a distributed database across trusted parties (court, police officials, etc.). The evidence log and ownership transfer details hold the details of evidence creation and deletion. The Evidence log acts as CoC. The results show that this method has an acceptable performance overhead. However, one authorized owner at a given point of time is good for ensuring valid CoC, but this causes a delay in the investigation process as only one among authorized entities can access the evidence.

Lone proposed two different models [109], [102] to ensure integrity of CoC using blockchain. In his first model [109], he implements a forensic chain using Ethereum and smart contracts. Details corresponding to the incident, i.e., the hash of digital evidence, location, date, and time, are recorded on the blockchain using a smart contract. A new block is added to the blockchain whenever digital evidence is accessed or transferred. The paper lacks the implementation of the proposed model. The author proposes Forensic-chain for CoC using Hyperledger composer in his later paper [102]. Hyperledger Composer is an open-source framework built on HLF blockchain infrastructure. Evidence creation, transfer, deletion, and display are captured on the blockchain. These details are used for auditing CoC. Hyperledger Caliper is used to evaluate the performance of the proposed model.

The paper LEChain [19] details supervising the entire evidence flow. It mainly emphasizes ensuring the privacy of the witness and juror by using randomizable signatures. The proposed model has been tested on the Ethereum blockchain network. The communication overhead and computational cost have been evaluated. However, the evidence is stored off-chain in the distributed database, but all stakeholders' transaction details performed on the evidential data, such as upload, access, and request, are stored on the blockchain.

Table 5.1: Digital Forensic Solutions based on Blockchain: Summary of strengths and weakness

Publication	Strength	Weakness
[101]	1. Addresses multi-stakeholder collusion. 2. Storage of cloud logs on-chain.	1. Huge volumes of cloud logs are replicated on block. 2. High storage overhead.
[102]	1. Ensures auditability of CoC. 2. Acceptable overhead in terms of resource utilization and throughput. 3. Used Hyperledger Caliper framework for performance evaluation.	1. Access to distributed storage system used to store the evidence is not discussed.
[105]	1. Data integrity is ensured even before the data is used as evidence. 2. Reduce dependency on CSP for integrity check.	1. Interoperability among CSPs. 2. Performance evaluation is done based on expected data size. 3. Role of investigator is not discussed.
[18]	1. Enables cloud auditing for the files stored on the cloud. 2. Data provenance is achieved with less overhead. 3. Performance overhead is independent of file size.	1. Not all cloud customers show interest to publish provenance data on public blockchain. 3. Dependency on provenance auditor.
[106]	1. Ensure evidence availability and integrity with reduced storage overhead.	1. Depends on trusted third party for storage.
[107]	1. Provenance of tamper-proof forensic artifacts submission list. 2. Negligible time overhead for group signature as it is executed only a couple of times.	1. Dependency on certification authority and process auditor. 2. Storage overhead depends on the on the no.of files in submission list.
[108]	1. The proposed model is synchronous this ensures consistency of data in CoC. 2. Acceptable storage overhead.	1. At a given point of time only one authorized user can access the evidence. 2. Increased communication overhead.
[109]	1. Ensures auditable CoC.	1. Lacks proof of concept. 2. Usage of public blockchain may not be recommended for forensics.
[19]	1. Ensures privacy of witness and Juror. 2. Usage of randomized signatures to ensure privacy. 3. Supervised evidence management.	1. Dependency on trusted authority. 2. Usage of public blockchain may not be recommended for forensics.

5.5 Proposed Model Investigation-Chain

This section discusses our proposed Investigation-Chain, its architecture, and workflow. Our model addresses three major problems of the existing system discussed in Section 5.1 i.e., 1. to ensure the integrity of evidence and avoid col-

lusion among multiple stakeholders involved in case investigation, 2. dependency on CSP, 3. delay in exchange of evidence legally.

We preserve evidential artifact cloud VM snapshot using Investigation-Chain. Cloud VM plays a significant role in cloud forensics. A VM instance involved in an incident can effectively restore the machine's state. It is a logical copy of the content on the virtual disk at a given time. We can analyze and access the files and folders stored on the VM by mounting the snapshot on a forensic workstation. The hash value of the snapshot is captured while collecting the evidence. It is used to ensure the integrity of the snapshot. In forensics, we can use VM snapshots for the following traces of evidence - hash list of files, recently modified files, deleted files recovery, list of malicious files and software, unusual start-up scripts, system logs, web activity, etc. The three main objectives of the proposed model are,

1. Ensure the integrity of the cloud VM snapshot.
2. Tamper-proof and transparent investigation model.
3. Availability of the snapshot to for the investigation.

To achieve the above objectives, we use permissioned blockchain, where all the stakeholders involved in the investigation are the blockchain network participants. The section 5.5.1 details the roles and responsibilities of each participant. The main idea is to update the investigation findings of each authenticated participant on the blockchain and ensure the availability of snapshot for analysis using the blockchain; this would help in transparent and quick court trials. Also, the transparent investigation procedure would increase public confidence in CSPs and the legal system. The architecture of the Investigation-Chain is shown in Figure 5.2.

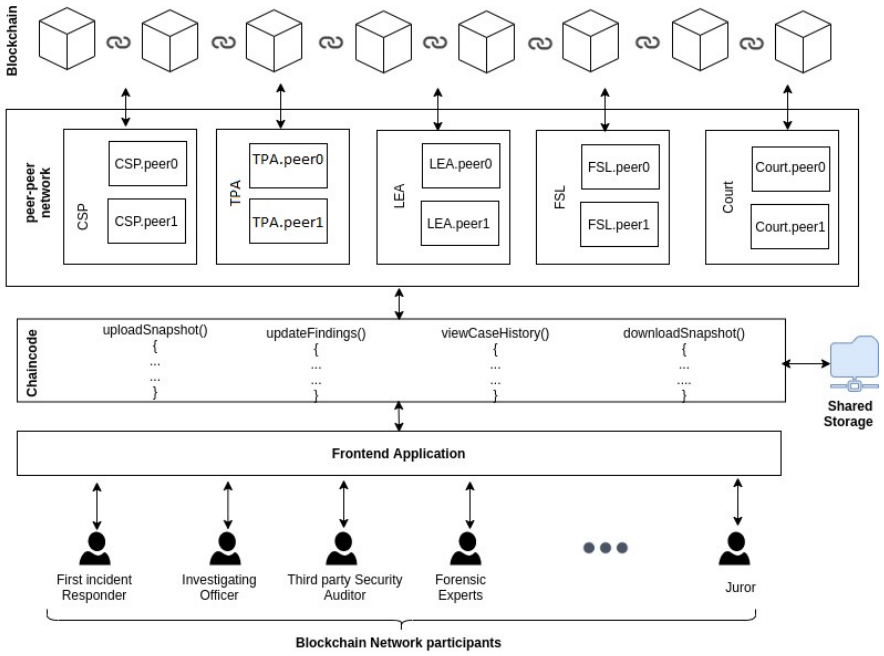


Figure 5.2: Architecture of Investigation-Chain

5.5.1 Blockchain Participants

In the event of the occurrence of a regular cyber incident, first incident responders, forensic investigators, prosecutor, defense, and court are the stakeholders involved in the investigation [102]. In the case of the cloud, we have additional stakeholders from CSP and trusted third parties. We analyzed the main stakeholders involved in handling cloud incidents by following the incident response guides of the leading cloud service providers like Amazon [65] and Google [110]. Further, we identified the organizations participating in the network and listed the participants from each organization below.

CSP (Organization 1)

- **First Incident Responder:** A first incident responder is responsible for identifying the incident first and reporting the incident to the next level. The member is responsible for capturing the snapshot of the suspected virtual instance, uploading the snapshot in a secured shared folder, capturing the snapshot's hash value, metadata of the snapshot, and updating all this information on the blockchain network.

The member should specify initial findings on the blockchain; this includes- 1. Target instance ID subscription ID or project ID. 2. Region of the target instance. 3. Time of occurrence of the incident. 4. Incident ticket no. 5. Intrusion detection system alerts, if any.

- **Security Analyst:** A security analyst is a peer member of the CSP organization on the blockchain. The member is responsible for downloading the snapshot from the shared path location on the blockchain, validating the snapshot's integrity by using the hash values recorded on the blockchain, analyzing the snapshot, and updating the member findings on the blockchain. The findings include- 1. Analysis of relevant logs, i.e., system logs, user activity logs, access logs, network flow logs. 2. Analysis of system configuration. 3. Analysis of boot history. 4. Identifying IP addresses.

- **Forensic Analyst:** A forensic analyst is a peer member of the CSP organization on the blockchain. The responsibilities of this member are similar to that of a security analyst. Additionally, this member examines the evidence forensically using forensic tools (e.g., FTK, Encase, TSK, Autopsy, etc.).

  The findings include- 1. Hash list of the files from the snapshot. 2. List of new or modified files. 3. List of files recovered from unallocated space. 4. Determine if any private keys are present. 5. Ignored security alerts generated by antivirus software. 6. Third-party hash lookups if any. 7. Timeline analysis, 8. List of unusual startup scripts. 9. List of suspicious files.

- **Incident Manager:** An incident manager is responsible for communicating the activities that correspond to the incident through the lifecycle of an incident. The Incident Manager is given the admin role for the CSP organization in the blockchain network. They are not involved much in the analysis of evidence; they are more involved in maintaining communication with other peer members in the network and auditing the transaction history.

LEA (Organization 2)

- IO: IO is a peer member of the LEA organization. An IO is responsible for the proceedings in the investigation. Every IO may not be a technical expert, but they are required to have basic knowledge of handling digital evidence. IO captures his findings on the blockchain.

The findings of an IO include- 1. Details of the complainant. 2. Analysis of physical crime scene location. 3. List of physical evidence found at the crime scene. 4. Details of the suspect, if any. 5. Details acquired during the interrogation. 6. Technical details such as metadata of VM snapshot and suspicious file.

- **Reporting Officer:** Reporting officer is responsible for communicating the activities corresponding to the incident through investigation. The reporting Officer is given the admin role for the CSP organization in the blockchain network. They are not much involved in analyzing evidence; instead, they maintain communication with other peer members in the network and audit the transactions.

Third Party Security Auditor (TPA) (Organization 3)

- **Security Auditor:** A security auditor is a peer member of the TPA organization on the blockchain. They are involved in analyzing evidence by the CSP for their expertise. The security auditor captures his findings on the blockchain. Their findings mainly involve assessing the cloud services if appropriate controls have been implemented.

Forensic Science Laboratory (FSL) (Organization 4)

- **Digital Forensic Analyst:** A forensic analyst is a peer member of FSLorganization. This member performs forensics on behalf of the LEA. As IO may or may not be technically sound, the forensic analyst does a complete forensic analysis of the snapshot. The member is responsible for downloading the snapshot from the shared path location on the blockchain, validating the integrity of the snapshot by using the hash values recorded on the blockchain, analyzing the snapshot, and recording his/her findings on the blockchain.

  Most of the findings updated on the blockchain are similar to those of the forensic analyst (CSP organization). However, these findings can be used to cross-validate the forensic analysis performed by the CSP.

Court(Organization 5)

- **Juror:** A juror is a member of the court organization in the blockchain network. The member holds the court trials and cross-validates the integrity

of the snapshot by using the hash values in the blockchain network. Audit the integrity of a snapshot and its findings stored on the blockchain network. They analyze the case presented by the prosecutor and defender and make appropriate judgments. This member is not involved in the snapshot analysis.

- **Prosecutor:** Prosecutor is a member of the blockchain network's court organization. During the court trials, they present the case against the accused. They download the snapshot, validate its integrity, and audit the findings updated by the other blockchain participants.
- **Defender:** A defender is a member of the court organization in the blockchain network. During the court trials, they defend the prosecution. They download the snapshot, validate its integrity, and audit the findings updated by the other blockchain participants. The member cross-validates the findings and checks if any inappropriate accusations have been made.

Based on their roles and responsibilities, the blockchain participants use the four main functions, i.e., 1. Upload snapshot, 2. Download snapshot, 3. Update findings, 4. View case history.

Upload snapshot uploads the snapshot in the shared folder so that all other blockchain participants can access the snapshot and download it. The first incident responder is responsible for uploading the snapshot. Algorithm 9 details the steps involved in uploading the snapshot. It takes the snapshot's metadata and the snapshot itself as input. The snapshot's metadata includes- the VM instance ID, the region where the instance is hosted, the path of the snapshot on the shared folder, the date and time of uploading the snapshot, etc. The algorithm initially computes the hash value of the snapshot to be uploaded H_{US}. We assign the hash value of the snapshot H_{US} to the caseID (i.e., $caseID = H_{US}$); this ensures that each caseID is unique and pulls the data corresponding to a specific snapshot. The algorithm checks if a caseID already exists by querying the blockchain. If the caseID does not exist, then a new caseID's value equivalent to H_{US} is created, and the snapshot is uploaded to the shared folder. Thus, by using caseID, we can add investigation findings and view case history specific to a snapshot from the blockchain. Also, it ensures that no snapshot is uploaded to the blockchain more than once, as caseID is equivalent to the hash value of the snapshot uploaded, so it avoids duplicates.

Algorithm 9 Upload Snapshot

```
procedure UploadSnapshot(snapshot, snapshot's metadata)
    H_US ← computeHash(snapshot) ▷ H_US: Hash value of snapshot to be uploaded
    caseID ← query(H_US)
    if caseID ≠ NULL then
        Case ID already exist
        return
    caseID = H_US
    Set case attributes using snapshot's metadata
    Upload snapshot to the shared folder
    return
```

Download snapshot is used to download the snapshot from the shared path location by other blockchain participants. Algorithm 10 details the steps involved in downloading the snapshot; it takes caseID as input. If the caseID requested by the blockchain participant exists, then the snapshot is downloaded. Once the blockchain participant downloads the snapshot, its hash value H_{DS} is computed again. We compare the hash value of the uploaded snapshot H_{US}, which is equivalent to caseID, with the hash value of the downloaded snapshot H_{DS} (i.e., if $caseID = H_{DS}$). If the hash values are equal, then we can say that the evidence has not been tampered with and can be analyzed for further investigation. By this, we ensure the integrity of the snapshot. The blockchain participants can now continue to analyze the downloaded snapshot and update their findings using the Update findings (Algorithm 11). Else, if the hash values are not found equal (i.e., $caseID \neq H_{DS}$), we can say that the snapshot has been tampered. This information is updated on the blockchain by using Update Findings to alert other blockchain participants. Thus making the model tamper-proof and transparent.

Algorithm 10 Download Snapshot

```
procedure DownloadSnapshot(caseID)
    if caseID ≠ NULL then
        Download snapshot
        Compute H_DS                ▷ H_DS: Hash value of downloaded snapshot
        if H_DS == caseID then
            Analyze the snapshot and update findings
            return
        else
            Update finding as Snapshot is tampered
            return
    else
        Case ID does not exist
        return
```

Update findings is used by the blockchain participants to add their investigation findings as transactions to the blockchain. Algorithm 11 details the steps involved in update findings; it takes caseID as input. Participant invokes the query method with case ID as an argument. The query method returns NULL if the snapshot with a specific case ID does not exist. If caseID exists, the participant submits the findings as a new transaction. The findings will be in descriptive format. As these findings are submitted as blockchain transactions, they are appended to the blockchain's immutable ledger; this ensures transparency, immutability, and integrity in the investigation.

Algorithm 11 Update Investigation Findings

```
procedure UpdateFindings(caseID)
    caseID ← query(caseID)
    if caseID ≠ NULL then
        Update the findings
    else
        Case ID does not exist
        return
```

View case history is used by the participants to view all the transactions corresponding to a specific caseID recorded on the blockchain ledger by other

participants. Algorithm 12 details the steps involved in viewing case history; it takes caseID as input. Participant invokes the query method with case ID as an argument. The query method returns NULL if the snapshot with a specific case ID does not exist. If the caseID exists, the ledger corresponding to a specific caseID is downloaded. The ledger entries are audited by cross-validating with the entries by other participants. By this, we can avoid collusion among participants to tamper with evidence. This ledger is audited for judgment during court trials. Thus, this ensures the auditability of the Investigation-Chain.

Algorithm 12 View Case History

```
procedure ViewHistory(caseID)
    caseID ← query(caseID)
    if caseID ≠ NULL then
        View case history
    else
        Case ID does not exist
        return
```

5.5.2 Investigation-Chain Workflow

The workflow diagram in Figure 5.3 shows a graphical overview of the Investigation-Chain. Step 1 in Figure 5.3 details the steps involved in uploading the snapshot by the first incident responder, and step 2 involves the details corresponding to downloading the snapshot, updating case finding, and accessing case history by other blockchain participants.

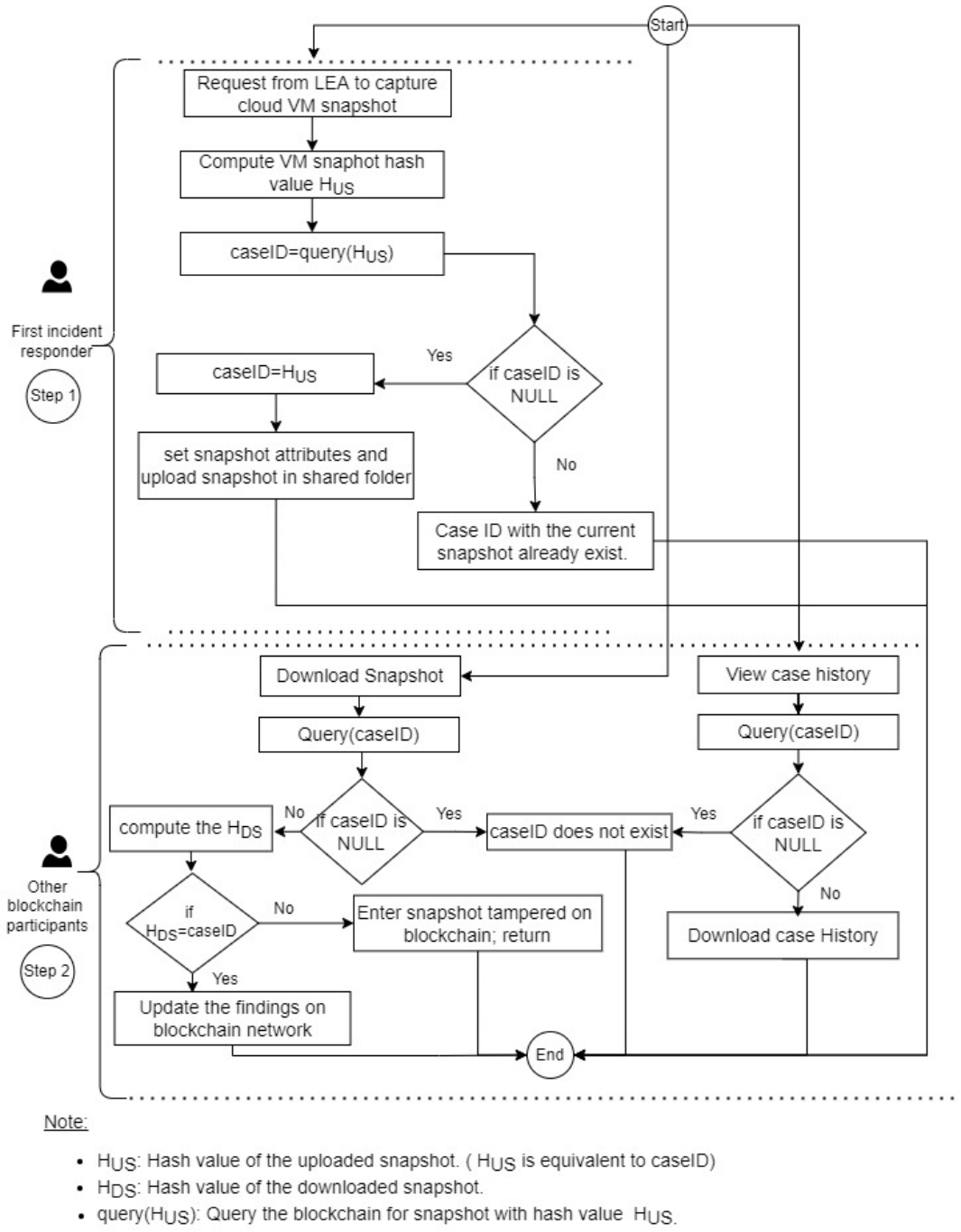


Figure 5.3: Operational workflow of Investigation-Chain

5.6 Proof of Concept

To validate the application of Investigation-Chain, we apply the model to a case. For simplicity, we considered a child pornography case. To simulate the cloud environment, we created a private cloud using Openstack (version: Wallaby) [111] on a single host machine. Further, we created an instance on the Openstack cloud using the Ubuntu cloud image (.qcow2). This instance is used as the target virtual instance in the case for analysis. For the blockchain network, we used HLF with five organizations, each organization with a single peer (5-organizations-1-peer).

5.6.1 Case Study (Child Pornography)

A victim of child sexual abuse approaches the local police station to file a complaint against the accused. An IO at the police station receives a complaint and interviews the victim. The IO then obtains a search warrant for the accused home to gather the evidence. The IO recovers a laptop, mobile phone, and other electronic storage media from the crime scene (i.e., the accused home). The IO uses forensic tools like FTK, Solo, Encase, etc, for data acquisition to capture the image of digital evidence found at the crime scene. Later, the digital evidence is seized and sent to the forensic lab for analysis.

Upon receiving the evidence, the forensic analyst analyzes the electronic devices seized. The analyst uses forensic tools like FTK, Encase, etc., to perform forensic analysis. The analysis is based on the nature of the incident, i.e., as the case is specific to child pornography, the analyst looks for images and video files, performs a keyword search, and checks if any photo morphing tools are present. In this case, to the surprise of the analyst, no illegal content specific to child sexual abuse is found on the suspect machine. Thus, the analyst further analyzes the accused web activities and finds that the accused has been using cloud services. The analyst suspects that the accused might be using cloud services for illegal activities. The forensic analyst now generates a report and summarizes the analysis. This report is shared with the IO. Based on the report, IO sends a legal notice to CSP and requests them to provide a snapshot of the virtual instance used by the accused to perform illegal activities. The notice issued consists of the details corresponding to the accused username/subscription ID, date, and time of the incident. The CSP uses these details to identify the target virtual instance. Investigation-Chain application for this scenario is detailed below.

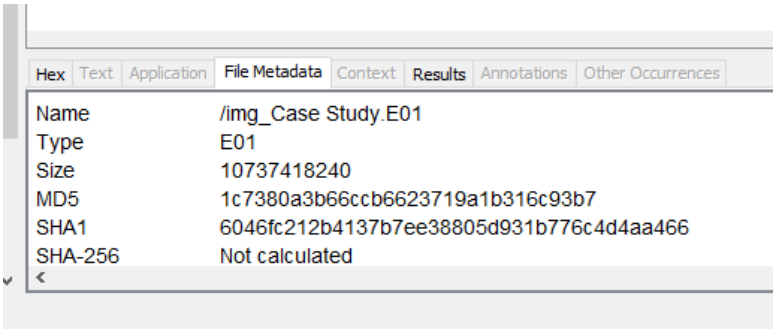


Figure 5.4: Hash value of snapshot computed using Autopsy tool

Step 1: Upon receiving the request from LEA, the first incident responder(at CSP) captures the snapshot of the target virtual instance hosted on the cloud. The first incident responder creates a new case ID and updates it with details corresponding to the incident on the blockchain. The details include the target instance ID, region, the path of the snapshot on the shared folder, and the hash value of the snapshot on the blockchain. Finally, submit the transaction with these details on the blockchain. When we query this transaction on the blockchain, it returns the submitted transaction in the JSON format, as shown below. Here, the key represents the caseID (hash of the snapshot).

```
 {
"Key": "1c7380a3b66ccb6623719a1b316c93b7",
"Record": {
  "participant": "First Incident Responder",
  "targetInstanceID": "ac21afab-4044-4fde-9bcd-28b82828b5d6",
  "region": "nova",
  "path": "/home/HyperLedger/snapshots/",
  "snap_hash": "1c7380a3b66ccb6623719a1b316c93b7",
}
```

Step 2: The other blockchain participants can download the snapshot and analyze it. To validate the integrity of the snapshot, the participant compares the hash values of the snapshot, i.e., validate if the hash value of the snapshot computed after downloading it from the shared path location is the same as the hash value of the snapshot updated on the block. We computed the hash value of the downloaded snapshot by using the Autopsy tool (Figure 5.4). It is found

that the hash value in Figure 5.4 is the same as the hash value of the snapshot on the blockchain (see JSON object above)

Step 3: The forensic analyst from the CSP organization now downloads the snapshot from the blockchain using the case ID. The downloaded snapshot is in '.VMDK' format. We used this file as input to the forensic tool. In this case, we used Autopsy, an open-source tool to perform forensic analysis. The forensic analyst analyses the reports generated by the Autopsy tool and looks for images and videos containing child porn content. The analyst finds an image named "child2.jpeg" related to child pornography. Figure 5.5 shows the screenshot of the Autopsy tool for metadata corresponding to the child2.jpeg image.

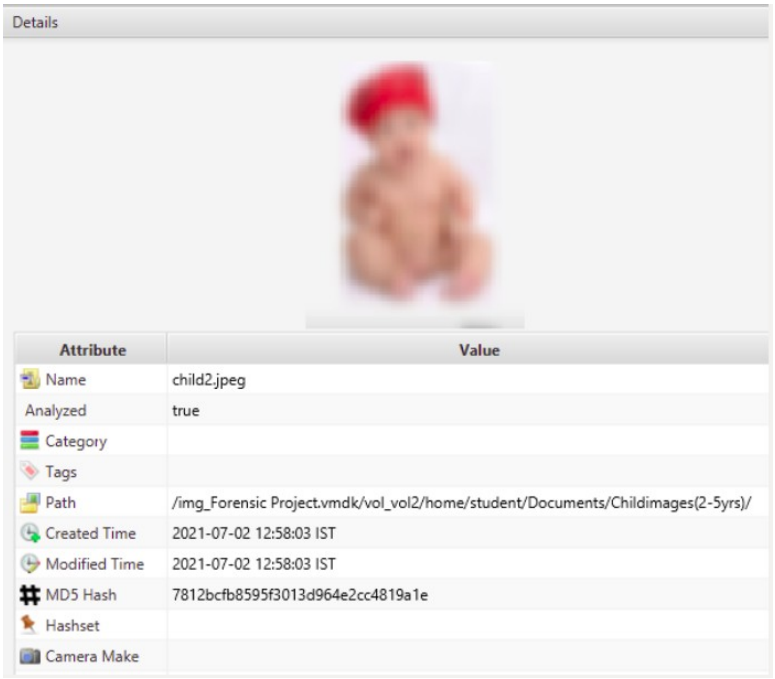


Figure 5.5: Cloud CSP instance image file findings using Autopsy tool

Step 4: The forensic analyst from the CSP organization now updates his findings regarding the child2.jpeg image on the blockchain; this includes the file name, its path on the snapshot, its hash value, file type, and date & time of creation. These details are updated on the blockchain as a new transaction, as shown below,


```
{
  "Key": "1c7380a3b66ccb6623719a1b316c93b7",
```

```
  "Record":  "{
    "snap_hash": "1c7380a3b66ccb6623719a1b316c93b7",
    "path": "/home/HyperLedger/snapshots/",
    "Findings": \"Name: child2.jpeg;
    File path: /img_Case Study.E01/vol_vol2/home/imgs/;
    Created Time : 2021-07-01 13:18:03 IST;
    Modified Time: 2021-07-01 13:18:03 IST ;
    MD5: 1a31589db2cfceae565eee7bd93a1371 ;
    MIME/type: image/png\",
    \"DateTime\":\"Thu 01 Jul 2021 06:47:12 PM IST\"}"
  }
}
```


Step 5: The forensic analyst from the FSL organization downloads the snapshot and validates its integrity, as mentioned in step 2. Further, the forensic analyst (from FSL) now performs the analysis using the forensic tool and updates his findings on the blockchain.

Step 6: Like the incident manager (CSP), reporting officer, juror, prosecutor, and defender, the other participants can view the ledger and cross-validate the updated findings. For example, the hash value of the image file ”child2.jpeg” updated by the forensic analyst (from CSP) can be compared with the hash value updated by the forensic analyst (from FSL). Similarly, every participating organization can view the entire ledger and validate the entire investigation proceedings.

5.7 Results and Discussion

5.7.1 Analysis of computational cost and communication overhead

We created a blockchain network on a laptop with 16 GB of RAM, Intel(R) Core(TM) i5-8265U CPU @1.60GHz, Ubuntu 18.04.6 LTS, and Visual Studio 2010. To set up the HLF network, we used the docker 20.10.7 and cloned the latest version of HLF from GitHub [103]. We created a blockchain network with five organizations and an orderer, each organization with one peer node. We generated the certificates for the participating nodes by using the certificate authorities.

To ensure privacy and confidentiality, we use HLF channels. Channels are the subnets among the network participants; only authenticated participants can join the channel and have private communication. Blockchain transactions are executed on these channels. The participants join the channel using the certificates created by the certificate authorities. We used Node.js to write the chaincode for the smart contract.

We analyzed the computational cost for the four main activities: uploading snapshots, updating findings, downloading the snapshots, and view case history. We compute the performance of Investigation-Chain by using the following terms,

- G_u- Generate the hash value of the snapshot uploaded.
- Q_h- Query the hash value of the snapshot on the blockchain.
- C_{id}- Create new case ID.
- U_s- Upload the snapshot in the shared folder.
- Q_{id}- Query the case ID.
- G_d- Generate the hash value of the downloaded snapshot.
- V_s- Validate the integrity of the snapshot by comparing the hash value of both uploaded and downloaded snapshots.
- D_s- Download the snapshot.
- A_{id}- Access the case History of a specific case ID from the blockchain.
- U_f- Update findings on the blockchain.

Uploading a snapshot involves- 1. Generating the hash value of the snapshot, 2. query the snapshot to see if it already exists on the blockchain network, 3. creating a new case ID, 4. uploading the snapshot. Thus, the time taken to upload the snapshot can be computed by $(G_u + Q_h + C_{id} + U_s)$, which takes approximately 2.6 seconds for a snapshot of size 40 MB. We have experimented with our model by varying the snapshot size from 40 MB to 500 MB. It is observed that as the size of the snapshot increases, the time required to upload the snapshot also increases. The graph in Figure 5.6 presents the time taken to upload a snapshot for each varying snapshot size.

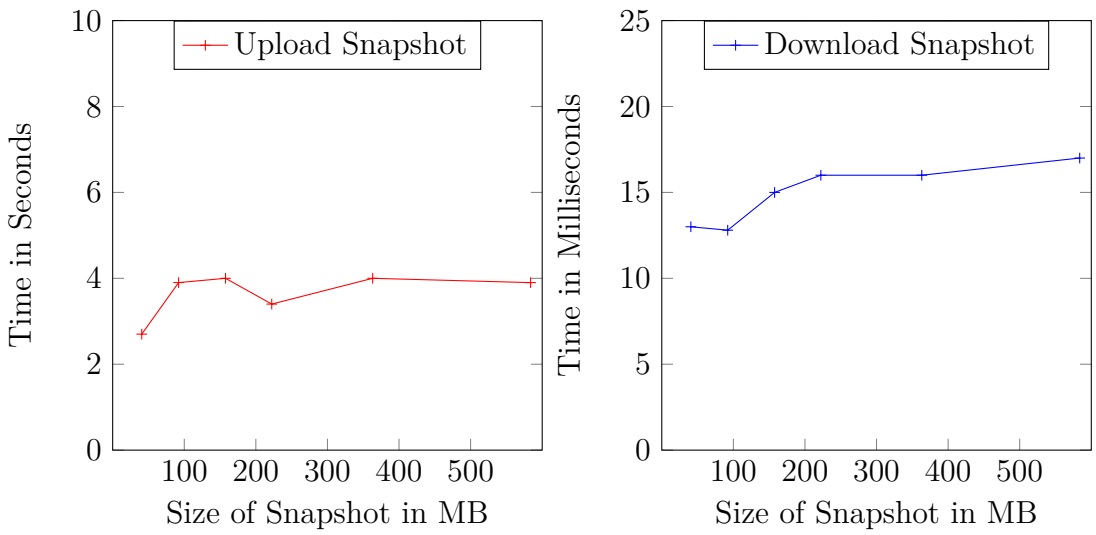


Figure 5.6: Time vs size of the snapshot

Downloading the snapshot involves- 1. Query the blockchain for the availability of the snapshot with the specific hash value. 2. Download the snapshot. 3. Generate the hash value of the downloaded snapshot. 4. Validate the integrity of the snapshot using hash values. Thus, the time taken to download the snapshot can be computed by $(Q_h + G_d + D_s + V_s)$, which takes approximately 13 milliseconds for the snapshot of size 40 MB. The graph in Figure 5.6 presents the time taken to download the snapshot for each varying snapshot's size from 40 MB to 500 MB. It is observed that there is not much variation in the time taken to download the snapshot, as the time variation is in milliseconds. The snapshot is available in the blockchain network's application server (i.e., shared folder); downloading takes less time than uploading a snapshot.

Other operations such as update findings and view case history involve- 1. Query the case ID, 2. update findings or view case history. Thus, the time taken to update finding can be computed by $(Q_{id} + U_f)$, which is approximately 18 milliseconds, and the time taken to view case history can be computed by $(Q_{id} + A_{id})$ which is approximately 11 milliseconds.

Table 5.2 lists the time taken by the Investigation chain for above discussed operations. It is seen that the time taken to upload the snapshot is longer compared to other operations as it involves copying the snapshot to the shared folder. The size of the snapshot considered for this experiment is 40MB. On the other

hand, the time taken to download the snapshot, access, and verification is comparatively acceptable.

Table 5.2: Operational cost in milliseconds

Operation	**Time (in ms)**
Upload snapshot	2679
Download snapshot	13.0
Update Findings	18.0
View History	11.0

We further evaluated the performance of Investigation-Chain using Hyperledger Caliper. It is a benchmark framework for performance evaluation of blockchain solutions [102]. Caliper generates reports on various performance indicators such as transactions per second (tps), latency, and resource utilization [112]. These reports help us determine the performance of blockchain under different scenarios and make choices based on user-specific requirements. We integrate the existing blockchain network with the caliper framework. Caliper provides interfaces to invoke the chaincode methods deployed on the blockchain network's peer nodes. We generate performance reports for Investigation-Chain by varying tps and network size from caliper interfaces. We computed blockchain performance indicators (generated by caliper), i.e., latency in seconds, throughput in tps, CPU utilization in percentage, memory utilization in MB, traffic in, and traffic out in KB.

We evaluated Investigation-Chain by varying the network size with 5-organization-1-peer, 4-organization-1-peer, and 3-organization-1-peer based on two functions, update findings and view the investigation history. These two functions involve submitting and querying transactions from the blockchain independent of storage services required to upload and download snapshots. We varied the transaction sending rate from 6 tps to 51 tps with an interval of 5. We repeated the experiment multiple times for each value of transaction send rate (tps) and computed the average values to reduce the error.

The Table 5.3, Table 5.4, and Table 5.5 shows average latency and throughput for 5-organizations-1-peer, 4-organizations-1-peer, and 3-organizations-1-peer, respectively. The results show that the latency gradually increases as we increase the transaction send rate. In contrast, the throughput rises to a point and progressively decreases after a certain transaction send rate (see Figure 5.7- 5.9).

The maximum throughput observed for 5-organizations-1-peer, 4-organizations-1-peer, and 3-organizations-1-peer is 14 tps, 15 tps, and 16 tps (approximately).

Table 5.3: Avg latency and throughput for a network with 5-organizations-1-peer

Send Rate(tps)	Max Latency(s)	Min Latency(s)	Avg Latency(s)	Throughput(tps)
6 tps	2.30	1.06	1.49	4.46
11 tps	2.1	0.63	1.25	6.63
16 tps	1.90	0.92	1.36	10.06
21 tps	2.97	0.73	1.42	11.88
26 tps	3.03	0.80	1.50	13.63
31 tps	3.78	0.89	1.88	11.4
36 tps	4.99	1.20	2.88	9.02
41 tps	5.69	1.20	3.35	8.18
46 tps	6.93	2.11	4.61	7.06
51 tps	8.53	3.59	6.07	6.19

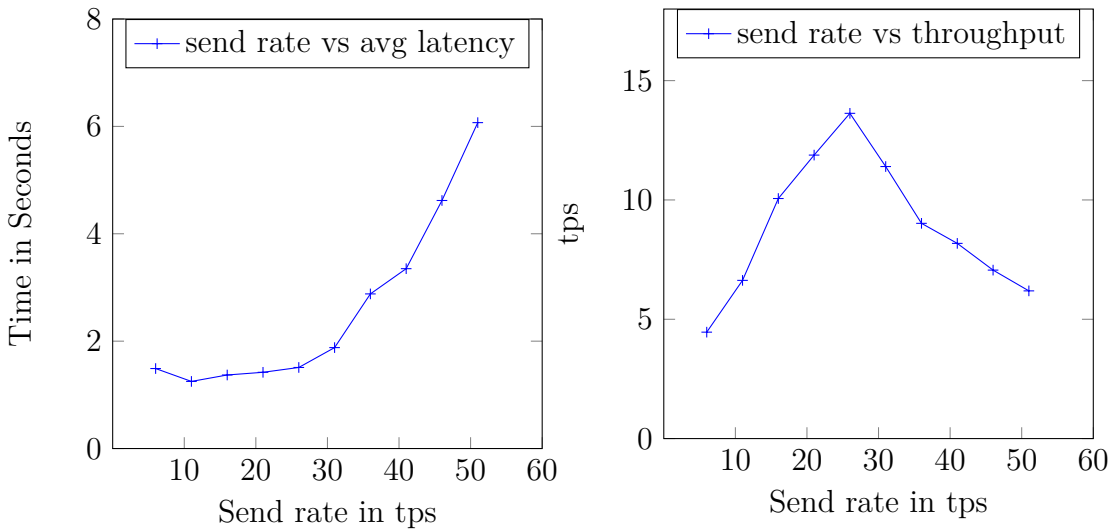


Figure 5.7: tps vs avg latency and throughput for 5-organizations-1-peer

Table 5.4: Avg latency and throughput for 4-organizations-1-peer

Send Rate	Max Latency(s)	Min Latency(s)	Avg Latency(s)	Throughput(tps)
6 tps	1.57	0.65	1.11	4.38
11 tps	1.81	0.48	1.04	6.83
16 tps	1.13	0.55	0.83	11.11
21 tps	2.14	0.51	1.02	12.35
26 tps	2.25	0.56	1.16	15.33
31 tps	3.47	0.53	1.40	13.94
36 tps	3.84	0.69	2.01	10.95
41 tps	4.77	1.02	2.70	9.85
46 tps	5.56	1.14	3.28	8.71
51 tps	6.68	1.54	4.08	7.85

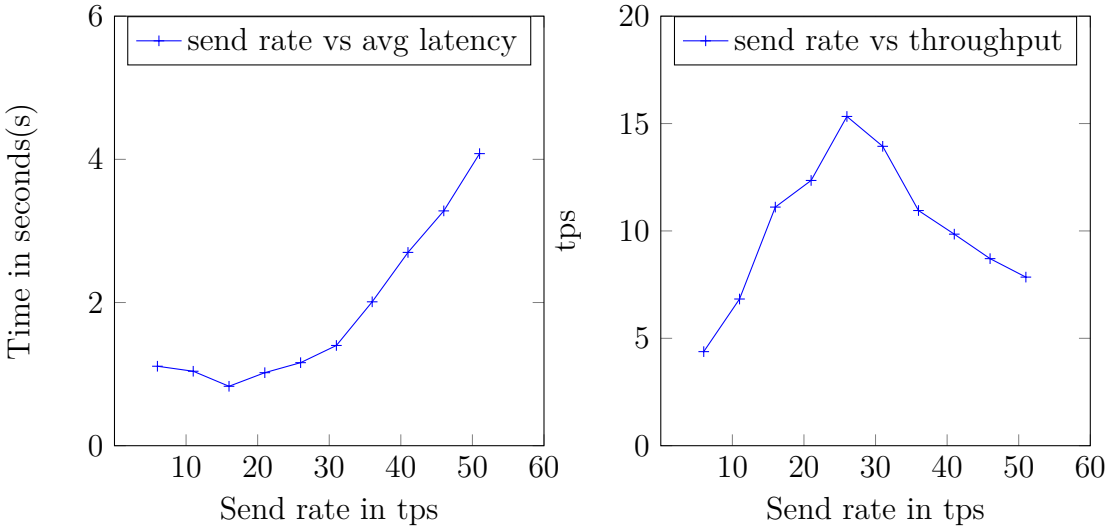


Figure 5.8: tps vs avg latency and throughput for 4-organizations-1-peer

Table 5.5: Avg latency and throughput for 3-organizations-1-peer

Send Rate	Max Latency(s)	Min Latency(s)	Avg Latency(s)	Throughput(tps)
6 tps	1.25	0.33	0.79	4.75
11 tps	1.36	0.52	0.92	7.16
16 tps	1.00	0.48	0.73	11.68
21 tps	1.83	0.56	1.02	12.88
26 tps	2.14	0.44	1.06	15.80
31 tps	3.45	0.43	1.67	10.24
36 tps	3.56	0.66	1.97	11.26
41 tps	4.33	0.62	2.25	10.78
46 tps	5.25	1.04	2.95	9.31
51 tps	5.87	1.43	3.62	8.91

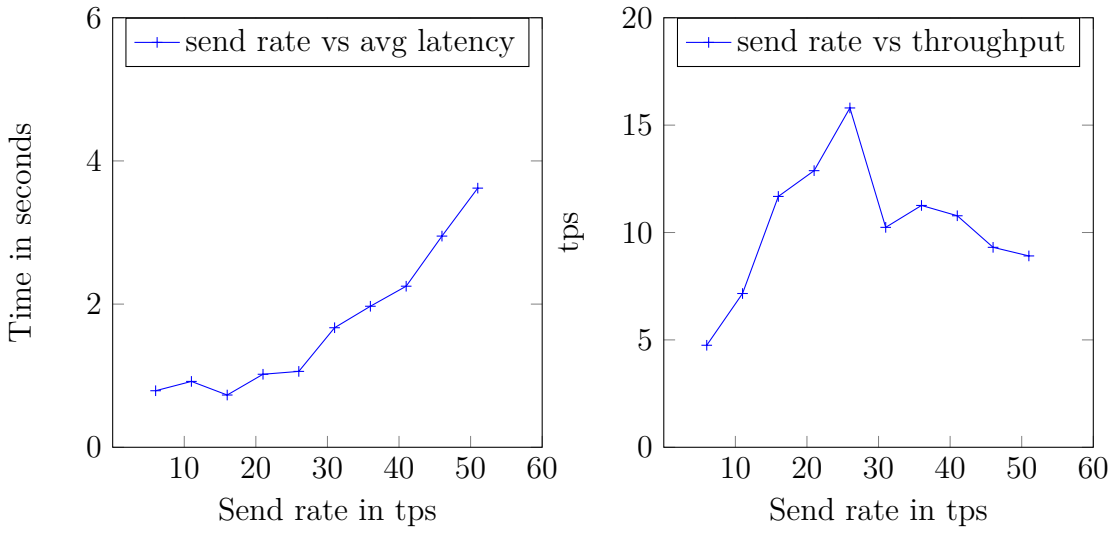


Figure 5.9: tps vs avg latency and throughput for 3-organizations-1-peer

Figure 5.10 compares the average latency, and Figure 5.11 compares the throughput for the above mentioned three network models. From Figure 5.10 and Figure 5.11, we observe that the throughput decreases and the average latency increases as the blockchain network size increases, which complies with the property of Hyperledger-based blockchain [102].

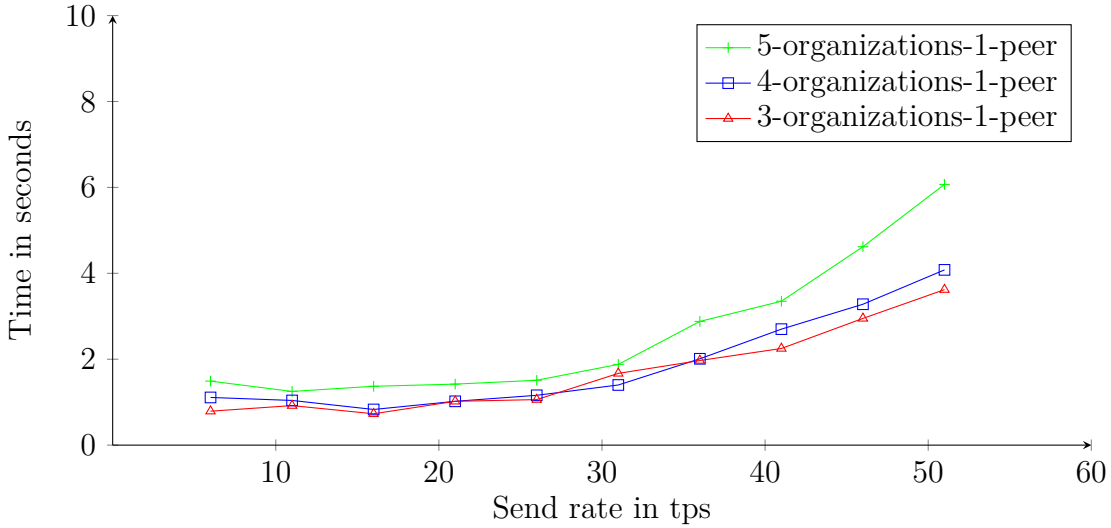


Figure 5.10: Average latency comparison for three network models

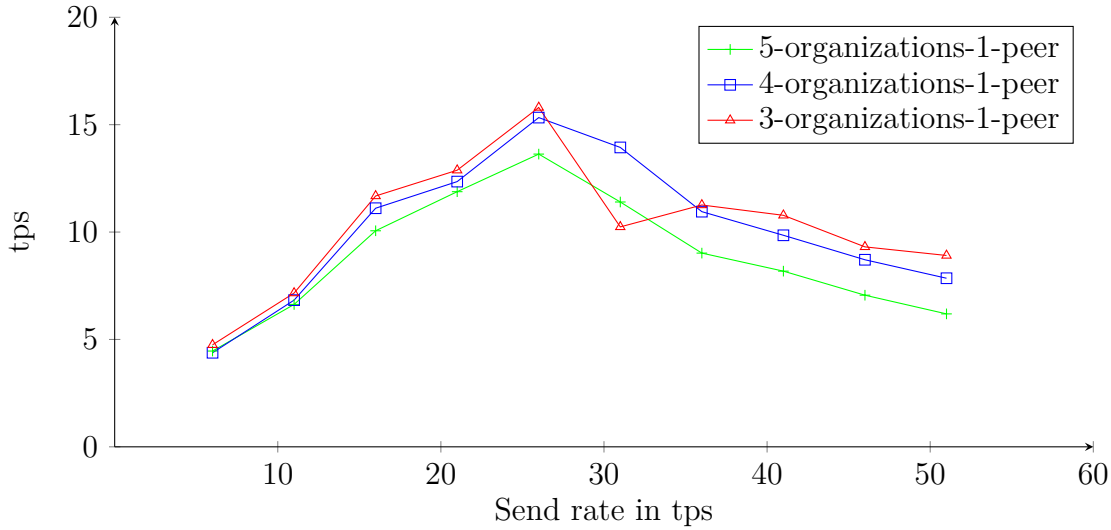


Figure 5.11: Throughput comparison for three network models

To analyze the communication overhead, we used Hyperledger Caliper and analyzed the resources consumed by each component involved in Investigation-Chain's blockchain network (5-organizations-1-peer). The table 5.6 details the resources used by each component of the proposed model for all transaction send rates. We initially repeated the test multiple times and computed the average CPU, average memory, and average traffic for every transaction send rate (for

each component). Further, we computed the average of CPU, memory, traffic in, and traffic out fields of the Table 5.6 for all transaction send rates (see Table 5.6).

Table 5.6: Average resource utilization for 5-organization-1-peer for all transaction send rate

Component Name	CPU%(avg)	Memory (avg) [MB]	Traffic In (avg) [KB]	Traffic Out (avg) [KB]
dev-peer0.csp.example.com	32.06	715.85	508.91	216.91
dev-peer0.lea.example.com	5.42	623.85	378.33	70.90
dev-peer0.tpa.example.com	0.02	587.21	1.72	0.69
dev-peer0.fsl.example.com	0.02	578.64	1.73	0.72
dev-peer0.court.example.com	0.11	586.39	1.70	0.68
peer0.csp.example.com	127.60	1282.07	2481.30	1301.67
peer0.lea.example.com	50.25	1167.11	1931.94	808.55
peer0.tpa.example.com	45.17	941.59	1508.74	445.23
peer0.fsl.example.com	45.08	1104.22	1513.00	439.71
peer0.court.example.com	44.98	1108.80	1510.06	444.26
orderer.example.com	5.13	682.12	1088.86	2295.95
csp_db	515.80	952.77	79.23	489.57
lea_db	46.65	901.40	57.84	29.37
tpa_db	49.90	712.42	68.92	38.28
fsl_db	46.21	715.53	68.73	38.45
court_db	48.61	710.96	68.80	38.17
ca_csp	4.08	109.74	11.09	23.39
ca_lea	4.05	224.22	11.06	23.29
ca_tpa	2.91	106.46	11.02	23.46
ca_fsl	1.79	105.42	10.96	23.43
ca_court	1.84	108.44	10.95	23.43
ca_orderer	0.08	214.11	0.81	0.00
cli	0.00	226.10	0.85	0.00

5.7.2 Comparative analysis of the recent research with Investigation-Chain

We present a comparative analysis of the recent research and our proposed model Investigation-Chain for the security of cloud evidential artifacts, such as integrity, auditability, privacy, confidentiality, transparency, and availability, in Table 5.7. The security elements listed in Table 5.7 are the common security aspects observed and mentioned explicitly across the recently published works. This comparative

analysis proves that Investigation-Chain fulfills all the essential security aspects compared to other research work.

- **Integrity** (Protection of block content from unauthorized modification):

  By its nature, Blockchain technology ensures an immutable ledger; we can ensure that the investigation findings and hash value of the snapshot recorded by blockchain participants are tamper-proof. Even if any peer tampers the ledger, it would not be able to convince all the other peers of the blockchain as the ledger is distributed throughout a network of independent peers. Hence, this makes this model tamper-proof and ensures integrity.

- **Auditability** (Ensuring the data on the blockchain can be used for audit trails): The investigation proceedings are available as transactions on the blockchain's immutable ledger. Blockchain participants can use the ledger to examine the transaction history(i.e., case investigation proceedings). Thus, the ledger allows blockchain participants to audit the case history and judge based on all transactions recorded on the blockchain.

- **Privacy** (Ensuring that only authorized members have access to the network): Maintaining data privacy is essential in the investigation, and only authentic users should participate in core functionalities. Investigation-Chain uses HLF with authorized blockchain participants. We used the certificate authorities to create the certificates to validate the identity of the blockchain participants. As all the participants are authenticated, privacy is ensured.

- **Confidentiality** (Protection of block content from unauthorized disclosure): HLF ensure confidentiality through channels. In Investigation-Chain, we created the channel and deployed our smart contract on it. The Blockchain transactions are executed on this channel. Only authenticated network participants who joined the channel using their certificates can access the ledger and smart contract. Thus, our model ensures confidentiality.

- **Transparency** (Ensuring all the peer members in the blockchain network can view all the transactions on the blockchain): The proposed model increases transparency in the investigation procedure by enabling the stakeholders to update their findings on the blockchain as transactions. Each

peer of the organization participating in the blockchain network has a local copy of the entire ledger, which the authenticated members of the organization can view. Thus, our model ensures transparency.

- **Availability** (Ensuring that the digital evidence is always available for analysis and examination by peer members): Availability of the evidence is ensured with reduced block size, i.e., without replicating the actual snapshot on the blockchain, which consumes a lot of network storage, we share the path of the shared folder and enable the user to download the snapshot. Thus, the evidence is available to all the blockchain participants, reducing the time required to access evidence, unlike in the existing system discussed in Section 5.1.

Table 5.7: Digital Forensic Solutions based on Blockchain: A Comparative Analysis of security elements

Publications	Blockchain type	Forensic artifact	Content on Block	Integrity	Auditability	Privacy	Confidentiality	Transparency	Availability
[101]	HLF	Cloud logs	Encrypted cloud logs	✓	X	✓	✓	✓	✓
[102]	Hyperledger Composer	CoC	History of ownership evidence creation, transfer & deletion details	✓	✓	✓	✓	✓	X
[105]	Permissioned blockchain	Cloud logs	Hash values of logs	✓	X	✓	✓	✓	X
[18]	Permissionless blockchain	Metadata of a file	hash values of metadata	✓	✓	X	✓	✓	X
[106]	Permissioned Blockchain	Files	Metadata of the file	✓	✓	✓	✓	✓	✓
[107]	Permissionless blockchain	Submission list of files	hash values of files in submission list	✓	✓	X	✓	✓	X
[108]	Private permissioned blockchain	CoC	History of ownership, evidence creation, transfer, & deletion details	✓	✓	✓	✓	✓	X
[109]	Ethereum	CoC	hash values of digital evidence, transfer & access request	✓	✓	X	X	✓	X
[19]	Ethereum	CoC	Evidence upload, access, request details	✓	✓	X	✓	✓	X
Investigation-Chain	HLF	cloud CSP Snapshot	Hash value of snapshot, shared folder path of the snapshot, investigation findings	✓	✓	✓	✓	✓	✓

5.8 Summary

Blockchain is a platform that provides multiple stakeholders who do not trust each other to share the information transparently and also ensure integrity, authenticity, and security by design. It is best suited for cloud forensics, where many stakeholders are involved. There is always a possibility that multiple stakeholders may collude among themselves to tamper with the digital evidence. This work presents a tamper-proof and transparent investigation model using the HLF framework. Investigation-Chain avoids multi-collusion problems among the stakeholders and ensures integrity, transparency, authenticity, privacy, and availability. We have seen that the Investigation-Chain enables the participating entities to upload CSP snapshots, push their findings by analysis of snapshots, and view the investigation findings updated by other participants using the immutable ledger.

Finally, we validated Investigation-Chain using a case study and Hyperledger caliper(a performance benchmark tool to evaluate the performance of a Hyperledger-based blockchain network). We compared our model with the computational cost of various operations such as upload, download, and access with other models. Further, computed the average latency, throughput, and resource utilized by varying the network size of Investigation-Chain by using Hyperledger caliper. The results show acceptable computational and communication overhead over the benefits offered by our model. Our analysis and experiments demonstrate a tamper-proof and transparent investigation model for the preservation of evidence and investigation proceedings.

Chapter 6

Conclusion and Future Work

In this chapter, concluding remarks are presented on the research work which is being carried out in this thesis. This chapter contains the summary of the contributions of this thesis and finally, a few glimpses of future research directions.

6.1 Summary of Contributions

Anti-forensics analyzes evidential artifacts to ensure their integrity and completeness of evidence so that it is admissible in a court of law. From our detailed survey of the existing cloud forensic procedures adopted by the leading CSPs, it was observed that the anti-forensics is ignored. In this thesis, we address one of the anti-forensic practices of artifact wiping. We focused on detecting wiping, recovery of wiped files from cloud VMs, and preservation of evidential artifacts like cloud VM snapshots.

To detect wiping on cloud VMs, we proposed two solutions using information theory metrics: a static approach WiDeJ and a dynamic approach WiDeS. In WiDeJ, we fetch the file system journal from the cloud VM snapshot and analyze the data blocks corresponding to a file to detect wiping. In WiDeS, we fetch the system-calls from cloud VMs using Sysdig and detect wiping by analyzing patterns in system-calls. Since there is no benchmark dataset available to check the accuracy of our model, we have used our own synthetic data. For the static approach, we have tested our model on the data set with 54 files (which includes 22 different file types). For WiDeS, we have tested it on our dataset with 93 processes (including both 70 benign and 15 wiping processes). In both cases, our models could determine wiping with 100% accuracy.

For the recovery of wiped files, we exploited the data recovery mechanism of the underlying file system of the cloud VM. In this thesis, we explored the journal-based file system Ext3 and the CoW-based file system BTRFS. We evaluated both file systems. We discussed recovery from the journal-based file system and its limitations. Further, we proposed using BTRFS in cloud VMs with our script preinstalled that keeps running in the background and logs the PA of the files upon their creation. Since BTRFS is based on the CoW principle, it contains multiple file versions. However, the reference for the previous version of the file is lost. Using ReWinD, we captured the PA of the file's previous versions. There are a few associated challenges with BTRFS, like node splitting and merging. We detailed the associated challenges and scope of recovery of files upon wiping the file.

Finally, we propose Investigation-Chain to preserve cloud VM and investigation proceedings using blockchain technology. This investigation model also addresses the challenges corresponding to cloud forensics, such as evidence integrity, transparency, immutability, and availability. We have identified the prime stakeholders in cloud incident investigation and defined their roles and responsibilities in detail. Further, we proposed the system architecture for cloud investigation by incorporating blockchain technology in cloud forensics. We also evaluated the performance of Investigation-Chain using the Hyperledger Caliper. The results show that our model's performance overhead is acceptable, considering the additional benefits offered.

6.1.1 List of Contributions

- An approach to detect file wiping on virtual machines using journals, data blocks, and information theory metrics.
- A novel approach to detect file wiping on virtual machines using the SoS and information theory metrics.
- An approach to restore the wiped files using file system journals.
- A novel approach to restore the wiped files using CoW-based file system BTRFS.
- A usecase for extending ReWinD for recovery of encrypted files using Gonnacry application.

- A big picture of cloud forensic approaches adopted by the leading CSPs.
- An investigation model to preserve evidential artifacts and capture investigation proceedings that is tamper-proof and transparent across the stakeholders involved in the investigation.

6.2 Future Work

In this thesis, we have limited the scope of our work to artifact wiping. However, we wish to extend our work to encryption. Encryption is also considered as a major anti-forensic approach, where the adversary tries to encrypt the file content, obfuscating the investigator to conduct a smooth investigation. Also, we found many similar characteristics between encryption and wiping (e.g., encrypted files contain random characters, like wiped files).

As we explore encryption, we also consider ransomware attacks due to their similarity with wiping attacks. This is because of ransomware behavioral characteristics like- mass file encryption, unusual file renaming activities, and frequent repetitive file access events are similar to wiping attacks. For example, to encrypt a single file, the ransomware performs three basic activities, i.e., 1) Opening the file, 2) reading the contents of the file, and 3) encrypting the file. Now, this pattern is repeated for multiple files that are available in the user space. This causes the same pattern of system-calls to be repeated multiple times, similar to wiper attacks. Thus, the proposed model, WiDeS, which uses system-calls and entropy, can be extended to detect ransomware attacks.

Also, in ReWinD, we have seen a usecase where we extend ReWinD to recover encrypted files affected by ransomware Gonnacry. In this direction, we propose to extend our work to determine the file system behavior under mass file encryption and the scope for recovering encrypted files when affected by ransomware attacks.

List of Publications

Journals

1. Sanda P, Pawar D, Radha V. **An insight into cloud forensic readiness by leading cloud service providers: a survey**. *Computing*. 2022 Apr 17:1-26. https://doi.org /10.1007/s00607-022-01077-2. [**Indexed: SCI, SCIE, SCOPUS, DBLP, UGC-CARE List(India)**]
   Status: Accepted and Published

2. Sanda P, Pawar D, Radha V. **Blockchain-based tamper-proof and transparent investigation model for cloud VMs**. *The Journal of Supercomputing*. 2022 May 25:1-29. https://doi.org/10.1007/s11227-022-04567-4. [**Indexed: SCIE, SCOPUS, UGC-CARE List(India)**]
   Status: Accepted and Published

3. Sanda P, Pawar D, Radha V. **ReWinD: Recovering Wiped and Deleted Files - An Anti-forensic Perspective** to *Forensic Science International: Digital Investigation*. [**Indexed: SCIE, SCOPUS**]
   Status: Under Review

Conference Proceedings

1. Sanda P, Pawar D, Radha V. **VM Anti-forensics: Detecting File Wiping Using File System Journals**. In *ICCET 2022*, International Conference on Computing in Engineering & Technology 2022 (pp. 497-508). Springer, Singapore. https://doi.org/10.1007/978-981-19-2719-5. [**Indexed: SCOPUS, EI, DBLP**]

2. Sanda P, Pawar D, Radha V. **WiDeS: Wiping Detection using System-calls - An Anti-forensic Resistant Approach**. In 2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (*TrustCom*) (pp. 1695-1703) **(Core ranking conference)**. https://doi.ieeecomputersociety.org/10.1109/TrustCom60117.2023.00231 [**Indexed: EI**]

Acronyms

AEP Asymptotic Equipartition Property.

ASCII American Standard Code for Information Interchange.

AWS Amazon Web Services.

BTRFS B-Tree File System.

CI Confidence Interval.

CoC Chain of Custody.

CoW Copy-on-Write.

CSP Cloud Service Provider.

DoD Department of Defense.

DTM Digital Tool Marks.

exFAT Extended File Allocation Table.

Ext Extended File System.

FAT File Allocation Table.

FN False Negative.

FP False Positive.

FS File System.

FSL Forensic Science Laboratory.

FTK Forensic Tool Kit.

GCP Google Cloud Platforms.

GDPR General Data Protection Regulation.

HIPPA Health Insurance Portability and Accountability Act.

HLF Hyperledger Fabric.

IaaS Infrastructure as a Service.

IO Investigating Officer.

IoT Internet of Things.

IRP I/O Request Packets.

JS Java Script.

LA Logical Address.

LEA Law Enforcement Agency.

MBR Master Boot Record.

MFT Master File Table.

NCRB National Crime Records Bureau.

NE Normalized Entropy.

NIST National Institute of Standards and Technology.

NTFS New Technology File System.

OES Octal Escape Sequences.

PA Physical Address.

PaaS Platform as a Service.

PHI Protected Health Information.

PII Personally identifiable information.

RAM Random Access Memory.

ReWinD Recovering Wiped and Deleted files.

SaaS Software as a Service.

SDK Software Development Kit.

SLA Service Level Agreement.

SOP Standard Operating Procedure.

SoS Sequence of System-calls.

TN True Negative.

TP True Positive.

TPA Third Party Security Auditor.

tps transactions per second.

TSK The Sleuth Kit.

VM Virtual Machine.

WiDeJ Wiping Detection using Journals.

WiDeS Wiping Detection using System-calls.

ZFS Zettabyte File System.

References

[1] Digambar Povar and VK Bhadran. **Forensic data carving**. In *Digital Forensics and Cyber Crime: Second International ICST Conference, ICDF2C 2010, Abu Dhabi, United Arab Emirates, October 4-6, 2010, Revised Selected Papers 2*, pages 137–148. Springer, 2011.

[2] Antonio Savoldi, Mario Piccinelli, and Paolo Gubian. **A statistical method for detecting on-disk wiped areas**. *Digital Investigation*, **8**(3-4):194–214, May 2012.

[3] Simson L Garfinkel. **Digital forensics research: The next 10 years**. *digital investigation*, **7**:S64–S73, 2010.

[4] Jignasa Sinha Mahender Singh Manral. **24% rise in cybercrime in 2022, 11% surge in economic offences: NCRB report**, December 2023.

[5] Acumen. **Digital Forensics Market Size - Global Industry, Share, Analysis, Trends and Forecast 2023 - 2032**, April 2023.

[6] Ryan Harris. **Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem**. *digital investigation*, **3**:44–49, 2006.

[7] Adeyinka Odebade, Thomas Welsh, Siyakha Mthunzi, and Elhadj Benkhelifa. **Mitigating anti-forensics in the cloud via resource-based privacy preserving activity attribution**. In *2017 Fourth International Conference on Software Defined Systems (SDS)*, pages 143–149. IEEE, 2017.

[8] Security Joes. **BiBi-Linux: A New Wiper Dropped By Pro-Hamas Hacktivist Group**, October 2023.

[9] Juan Andrés Guerrero-Saade. **HermeticWiper— New Destructive Malware Used in Cyber Attacks on Ukraine**. *Sentinel Labs*, 2022.

[10] Shams Zawoad and Ragib Hasan. **Cloud forensics: a meta-study of challenges, approaches, and open problems**. *arXiv preprint arXiv:1302.6312*, 2013.

[11] Gartner. **Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach $679 Billion in 2024**, November 2023.

[12] Thales. **CLOUD ASSETS THE BIGGEST TARGETS FOR CYBERATTACKS, AS DATA BREACHES INCREASE**, Jul 2023.

[13] Rawan Abdulaziz Al-Mulhim, Lama Adnan Al-Zamil, and Fay Mohammed Al-Dossary. **Cyber-attacks on Saudi Arabia environment**. *International Journal of Computer Networks and Communications Security*, **8**(3):26–31, 2020.

[14] Sanjeev Das, Yang Liu, Wei Zhang, and Mahintham Chandramohan. **Semantics-based online malware detection: Towards efficient real-time protection against malware**. *IEEE transactions on information forensics and security*, **11**(2):289–302, 2015.

[15] Smita Naval, Vijay Laxmi, Muttukrishnan Rajarajan, Manoj Singh Gaur, and Mauro Conti. **Employing program semantics for malware detection**. *IEEE Transactions on Information Forensics and Security*, **10**(12):2591–2604, December 2015.

[16] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. **Intrusion detection using sequences of system calls**. *Journal of computer security*, **6**(3):151–180,JCS–980109, 1998.

[17] Ohad Rodeh, Josef Bacik, and Chris Mason. **BTRFS: The Linux B-tree filesystem**. *ACM Transactions on Storage (TOS)*, **9**(3):1–32, 2013.

[18] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. **Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability**. In *2017 17th*

IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pages 468–477. IEEE, 2017.

[19] Meng Li, Chhagan Lal, Mauro Conti, and Donghui Hu. **LEChain: A blockchain-based lawful evidence management scheme for digital forensics**. *Future Generation Computer Systems*, **115**:406–420, 2021.

[20] Simson Garfinkel. **Anti-forensics: Techniques, detection and countermeasures**. In *2nd International Conference on i-Warfare and Security*, **20087**, pages 77–84, 2007.

[21] Kevin Conlan, Ibrahim Baggili, and Frank Breitinger. **Anti-forensics: Furthering digital forensic science through a new extended, granular taxonomy**. *Digital investigation*, **18**:S66–S75, 2016.

[22] Hussein Majed, Hassan N Noura, and Ali Chehab. **Overview of Digital Forensics and Anti-Forensics Techniques**. In *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–5. IEEE, 2020.

[23] Anu Jain and Gurpal Singh Chhabra. **Anti-forensics techniques: An analytical review**. In *2014 Seventh International Conference on Contemporary Computing (IC3)*, pages 412–418. IEEE, 2014.

[24] Bryan Sartin. **ANTI-Forensics–distorting the evidence**. *Computer Fraud & Security*, **2006**(5):4–6, 2006.

[25] Eoghan Casey. **Practical approaches to recovering encrypted digital evidence**. *International Journal of Digital Evidence*, 2002.

[26] Eoghan Casey, Geoff Fellows, Matthew Geiger, and Gerasimos Stellatos. **The growing impact of full disk encryption on digital forensics**. *Digital Investigation*, **8**(2):129–134, 2011.

[27] Adedayo M Balogun and Shao Ying Zhu. **Privacy impacts of data encryption on the efficiency of digital forensics technology**. *arXiv preprint arXiv:1312.3183*, 2013.

[28] Aniello Castiglione, Alfredo De Santis, and Claudio Soriente. **Taking advantages of a disadvantage: Digital forensics and steganography using document metadata**. *Journal of Systems and Software*, **80**(5):750–764, 2007.

[29] Merrill Warkentin, Ernst Bekkering, and Mark B Schmidt. **Steganography: Forensic, security, and legal issues**. *Journal of Digital Forensics, Security and Law*, **3**(2):2, 2008.

[30] Gary C Kessler. **Anti-forensics and the digital investigator**. 2007.

[31] Kamal Dahbur and Bassil Mohammad. **The anti-forensics challenge**. In *Proceedings of the 2011 International Conference on Intelligent Semantic Web-Services and Applications*, pages 1–7, 2011.

[32] Derek Manky. **FortiGuard Labs Reports Destructive Wiper Malware Increases Over 50%**, 2023.

[33] Rayed AlHarbi, Ali AlZahrani, and Wasim Ahmad Bhat. **Forensic analysis of anti-forensic file-wiping tools on Windows**. *Journal of forensic sciences*, **67**(2):562–587, October 2022.

[34] Graeme Horsman. **Digital tool marks (DTMs): a forensic analysis of file wiping software**. *Australian Journal of Forensic Sciences*, **53**(1):96–111, 2021.

[35] Dabin Joo, Jiwon Lee, and Doowon Jeong. **A reference database of Windows artifacts for file-wiping tool execution analysis**. *Journal of Forensic Sciences*, 2023.

[36] Kyoung Jea Park, Jung-Min Park, Eun-jin Kim, Chang Geun Cheon, and Joshua I James. **Anti-forensic trace detection in digital forensic triage investigations**. *Journal of Digital Forensics, Security and Law*, **12**(1):8, March 2017.

[37] Christopher Swenson, Raquel Phillips, and Sujeet Shenoi. **File system journal forensics**. In *IFIP International Conference on Digital Forensics*, pages 231–244. Springer, 2007.

[38] Zachary Peterson and Randal Burns. **Ext3cow: A time-shifting file system for regulatory compliance**. *ACM Transactions on Storage (TOS)*, **1**(2):190–212, 2005.

[39] Josiah Dykstra and Alan T Sherman. **Understanding issues in cloud forensics: two hypothetical case studies**. *UMBC Computer Science and Electrical Engineering Department*, 2011.

[40] Keyun Ruan, Joe Carthy, Tahar Kechadi, and Ibrahim Baggili. **Cloud forensics definitions and critical criteria for cloud forensic capability: An overview of survey results**. *Digital Investigation*, **10**(1):34–43, 2013.

[41] Ben Martini and Kim-Kwang Raymond Choo. **Cloud forensic technical challenges and solutions: A snapshot**. *IEEE Cloud Computing*, **1**(4):20–25, 2014.

[42] NIST Cloud Computing Forensic Science Working Group et al. **Nist cloud computing forensic science challenges**. Technical report, National Institute of Standards and Technology, 2014.

[43] Martin Herman, Michaela Iorga, Ahsen Michael Salim, Robert H Jackson, Mark R Hurst, Ross Leo, Richard Lee, Nancy M Landreville, Anand Kumar Mishra, Yien Wang, et al. **NIST Cloud Computing Forensic Science Challenges**. Technical report, National Institute of Standards and Technology, 2020.

[44] Bharat Manral, Gaurav Somani, Kim-Kwang Raymond Choo, Mauro Conti, and Manoj Singh Gaur. **A Systematic Survey on Cloud Forensics Challenges, Solutions, and Future Directions**. *ACM Computing Surveys (CSUR)*, **52**(6):1–38, 2019.

[45] Ameer Pichan, Mihai Lazarescu, and Sie Teng Soh. **Cloud forensics: Technical challenges, solutions and comparative analysis**. *Digital investigation*, **13**:38–57, 2015.

[46] Keyun Ruan, Joe Carthy, Tahar Kechadi, and Mark Crosbie. **Cloud forensics**. In *IFIP International Conference on Digital Forensics*, pages 35–46. Springer, 2011.

[47] Ahmed Alenezi, Hany F Atlam, and Gary B Wills. **Experts reviews of a cloud forensic readiness framework for organizations**. *Journal of Cloud Computing*, **8**(1):11, 2019.

[48] Deevi Radha Rani and G Geetha Kumari. **A framework for detecting anti-forensics in cloud environment**. In *2016 International Conference on Computing, Communication and Automation (ICCCA)*, pages 1277–1280. IEEE, 2016.

[49] Prasad Purnaye and Vrushali Kulkarni. **A comprehensive study of cloud forensics**. *Archives of Computational Methods in Engineering*, **29**(1):33–46, 2022.

[50] Bala Raj, Gill Bob, Smith Dennis, Wright David, and Kevin Ji. **Magic Quadrant for Cloud Infrastructure as a Service, Worldwide**, sep 2020.

[51] Amazon. **Simplify Security Incident Response and Digital Forensics on AWS**, 2020.

[52] Morgan Arundell Barry Conway. **Automated Forensics and Incident Response on AWS**, May 2019.

[53] Microsoft. **Computer Forensics in Azure**, 2020.

[54] Frank Simorjay Ben Ridgway. **Microsoft Azure Security Response in the Cloud**, April 2016.

[55] Sami Zuhuruddin. **Cloud Forensics**, Jul 2018.

[56] Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Mustapha Aminu Bagiwa, Muhammad Shiraz, Samee U Khan, Rajkumar Buyya, and Albert Y Zomaya. **Cloud log forensics: Foundations, state of the art, and future directions**. *ACM Computing Surveys (CSUR)*, **49**(1):1–42, 2016.

[57] Amazon. **Amazon CloudWatch Logs User Guide**, 2020.

[58] Amazon. **Centralized Logging**, Dec 2020.

[59] Wren Brian, Rita, Coulter David, and Mutha Piyush. **Azure Monitor overview**, Nov 2019.

[60] Google. **CLOUD LOGGING DOCUMENTATION**, 2020.

[61] Google. **Google Available Logs**, Jan 2020.

[62] Amazon. **How S3 Object Lock works**, 2020.

[63] Microsoft. **Corporate Social Responsibility**, September 2018.

[64] Google. **Google Transparency Report**, February 2021.

[65] Amazon. **AWS Security Incident Response Guide**, June 2020.

[66] Google. **Identity and Access Management**, March 2021.

[67] Amazon. **Amazon Law Enforcement Guidelines**, 2021.

[68] Microsoft. **Corporate Social Responsibility**, 2021.

[69] Google. **Google Transparency Report**, 2021.

[70] Wasim Ahmad Bhat, Ali AlZahrani, and Mohamad Ahtisham Wani. **Can computer forensic tools be trusted in digital investigations?** *Science & Justice*, **61**(2):198–203, 2021.

[71] Tim Fisher. **Data Sanitization Methods**, 2021.

[72] Vijayan Prabhakaran, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. **Analysis and Evolution of Journaling File Systems.** In *USENIX Annual Technical Conference, General Track*, **194**, pages 196–215, 2005.

[73] Gregorio Narváez. **Taking advantage of Ext3 journaling file system in a forensic investigation**. *SANS Institute Reading Room*, 2007.

[74] Dong Bin Oh, Kyung Ho Park, and Huy Kang Kim. **De-Wipimization: Detection of data wiping traces for investigating NTFS file system**. *Computers & Security*, **99**:102034, 2020.

[75] Uma Kumar, Vinod Kumar, and J N Kapur. **Normalized measures of entropy**. *International Journal Of General System*, **12**(1):55–69, 1986.

[76] Joao BD Cabrera, Lundy Lewis, and Raman K Mehra. **Detection and classification of intrusions and faults using sequences of system calls**. *Acm sigmod record*, **30**(4):25–34, 2001.

[77] Andrea Continella, Alessandro Guagnelli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi. **ShieldFS: A Self-healing, Ransomware-aware Filesystem**. In *Proceedings of the 32nd Annual Computer Security Applications Conference*. ACM, 2016.

[78] LORIS DEGIOANNI. **The Fascinating World of Linux System Calls**, dec 2014.

[79] Calvin B Paul. *Entropy-based file type identification and partitioning*. PhD thesis, Monterey, California: Naval Postgraduate School, 2017.

[80] Michael Kerrisk. **Linux/UNIX System Programming Essentials**, jun 2023.

[81] Wasim Ahmad Bhat and Mohamad Ahtisham Wani. **Forensic analysis of B-tree file system (Btrfs)**. *Digital Investigation*, **27**:57–70, 2018.

[82] Devyani Gurjar and Satish S Kumbhar. **A review on performance analysis of ZFS & BTRFS**. In *2019 International Conference on Communication and Signal Processing (ICCSP)*, pages 0073–0076. IEEE, 2019.

[83] Jan-Niclas Hilgert, Martin Lambertz, and Shujian Yang. **Forensic analysis of multiple device BTRFS configurations using The Sleuth Kit**. *Digital Investigation*, **26**:S21–S29, 2018.

[84] Fedora. **Fedora Workstation Documentation Disk Configuration**, 2024. Accessed on Jan 22, 2024.

[85] openSUSE. **SUSE Linux Enterprise Server Release Notes**, 2022. Accessed on Jan 22, 2024.

[86] Facebook. **Improving machine learning iteration speed with faster application build and packaging**, 2024. Accessed on Jan 28, 2024.

[87] Netgear. **Netgear Software Manual**, 2019. Accessed on Jan 28, 2024.

[88] Rockstor. **Linux Btrfs NAS Server**, 2024. Accessed on Jan 28, 2024.

[89] Synology. **How Btrfs protects your company's data**, 2024. Accessed on Jan 28, 2024.

[90] Docker. **Use the BTRFS storage driver**, 2024. Accessed on Jan 30, 2024.

[91] LXD. **BTRFS driver in LXD**, 2024. Accessed on Jan 30, 2024.

[92] Kevin D Fairbanks. **A technique for measuring data persistence using the ext4 file system journal**. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, **3**, pages 18–23. IEEE, 2015.

[93] Howard Powell. **ZFS and Btrfs: a quick introduction to modern filesystems**. *Linux Journal*, **2012**(218):5, 2012.

[94] btrfs readthedocs. **BTRFS Documentation**, 2023. Accessed on Jan 27, 2024.

[95] btrfs wiki. **BTRFS Wiki**, 2017. Accessed on Jan 19, 2024.

[96] Mohamad Ahtisham Wani, Wasim Ahmad Bhat, and Ali Dehghantanha. **An analysis of anti-forensic capabilities of B-tree file system (Btrfs)**. *Australian Journal of Forensic Sciences*, **52**(4):371–386, 2020.

[97] btrfs readthedocs. **BTRFS Documentation**, 2023. Accessed on Jan 27, 2024.

[98] Daniel. **Understanding btrfs internals part 3**, 2020. Accessed on Jan 27, 2024.

[99] Karel Zak. **btrfs-progs**, 2024. Accessed on Jan 27, 2024.

[100] Thomas K Dasaklis, Fran Casino, and Constantinos Patsakis. **Sok: Blockchain solutions for forensics**. *arXiv preprint arXiv:2005.12640*, 2020.

[101] Sagar Rane and Arati Dixit. **BlockSLaaS: Blockchain assisted secure logging-as-a-service for cloud forensics**. In *International Conference on Security & Privacy*, pages 77–88. Springer, 2019.

[102] Auqib Hamid Lone and Roohie Naaz Mir. **Forensic-chain: Blockchain based digital forensics chain of custody with PoC in Hyperledger Composer**. *Digital Investigation*, **28**:44–55, 2019.

[103] Hyperledger. **Hyperledger Fabric**, 2020.

[104] Omi Akter, Arnisha Akther, Md Ashraf Uddin, and Md Manowarul Islam. **Cloud Forensics: Challenges and Blockchain Based Solutions [J]**. *International Journal of Modern Education and Computer Science*, **10**(8):1–12, 2020.

[105] Jun Hak Park, Jun Young Park, and Eui Nam Huh. **Block chain based data logging and integrity management system for cloud forensics**. *Computer Science & Information Technology*, **149**, 2017.

[106] Zhihong Tian, Mohan Li, Meikang Qiu, Yanbin Sun, and Shen Su. **Block-DEF: A secure digital evidence framework using blockchain**. *Information Sciences*, **491**:151–165, 2019.

[107] Yong Zhang, Songyang Wu, Bo Jin, and Jiaying Du. **A blockchain-based process provenance for cloud forensics**. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 2470–2473. IEEE, 2017.

[108] Silvia Bonomi, Marco Casini, and Claudio Ciccotelli. **B-coc: A blockchain-based chain of custody for evidences management in digital forensics**. *arXiv preprint arXiv:1807.10359*, 2018.

[109] Auqib Hamid Lone and Roohie Naaz Mir. **Forensic-chain: Ethereum blockchain based digital forensics chain of custody**. *Scientific and Practical Cyber Security Journal*, **1**(2):21–27, 2018.

[110] Google. **Data incident response process**, September 2018.

[111] Openstack. **Installation guide**, 2021.

[112] Hyperledger Caliper. **Hyperledger Caliper Getting Started**, 2021.

[113] Amazon. **Collecting Metrics and Logs from Amazon EC2 Instances and On-Premises Servers with the CloudWatch Agent**, 2020.

[114] Wren Brian, Coulter David, Stolz Henry, and Dhanwada Swathi. **rview of Azure Monitor agents**, Jan 2021.

[115] David Coulter Brian Wren. **Log Analytics agent overview**, Jan 2021.

[116] Azure. **Azure Log Analytics**, 2021.

[117] Microsoft. **Digital Evidence Capture**, oct 2020.

[118] Google. **Google Cloud's operations suite**, 2020.

[119] Google. **CLOUD MONITORING DOCUMENTATION**, 2020.

[120] Google. **Google Rapid Response**, Oct 2020.

Appendix A

Cloud Forensic Workflows of Leading CSP

A.1 AWS Workflow

In AWS the cloud incidents are initially classified as service domain incidents and infrastructure domain incidents. Service domain incidents refer to the incidents associated with AWS services, and infrastructure incidents are associated with the physical or the virtual infrastructure supporting AWS services.

For service domain incidents, the logs corresponding to the specific service are collected and analyzed using Amazon Cloud Watch. Amazon, for effective log analysis, has implemented Amazon CloudWatch, which centralizes the logs from different sources and enables automation of alerts based on the occurrence of specific events [57] [58]. Cloudwatch collects the monitoring data in the form of logs, metrics, events and provides a unified view of AWS resources. It collects, analyzes, and displays logs and log metrics on a single dashboard. Amazon CloudWatch Agent [113] is used to collect the logs from AWS and on-premise resources running either Linux or Windows Server. The metrics collected by the CloudWatch agent are stored and viewed on AWS CloudWatch. The metrics collected by the CloudWatch agent are considered customized metrics and are billable. Amazon Kinesis is used by AWS to perform big data analytics on cloud logs [65].

Other resources used for forensics on AWS platform include - 1. Amazon step function - these are used for automating decision-based workflow. 2. Amazon

Machine Images (AMI) - these are used to launch new instances with the required configuration. 3. Amazon storage services (AWS S3, AWS S3 Glacier) - these services are used for archiving the evidence for the long term. 4. AWS SSM (System Management Agent) - this service is used to manage instances and automate operational tasks on AWS or on-premise resources. 5. Amazon Simple Notification Service (AWS SNS) - this service is used for sending notifications to the incident response team for forensic process updations. Forensics at AWS is performed by incident response teams and security teams.

Forensics at AWS is automated by using AWS step functions. The following step functions are used to perform forensics at AWS - 1. triage step function, 2. disk step function, 3. processing step function. The *Triage step function* determines the type of acquisition needed, i.e., either disk acquisition or memory acquisition. This decision is taken based on the metadata captured by the malicious incident. For disk forensics, the triage step function invokes the disk step function and shares the target VM instance ID for which the snapshot is to be captured.

Disk step function initially protects the targeted instance from getting terminated (i.e., deletion of an instance, its associated data disk, and volumes) by enabling termination protection of instance. Later, the instance is isolated by changing the security group of the instance to restricted mode. The security group of an instance controls in and out network traffic to the instance. Further, detach the instance from the auto-scaling group by deregistering the instance from the Elastic Load Balancer (ELB). Next, create the snapshot of the target instance and its attached disk volumes and generate the hash values using SHA-256, MD5 algorithms. Create a new AWS EC2 instance in AWS security teams' Virtual Private Cloud (VPC) by using forensic workstation AMI. Attach the snapshot and disk volumes to forensic workstation AMI. Install AWS SSM on the forensic instance. This is used in the processing step function to automate data preprocessing for analysis. Finally, the disk step function triggers the processing step function.

The processing step function automates the basic investigation needed. The AWS SSM installed on the forensic instance prepares the data on the snapshot for analysis by using forensic tool, i.e., for Windows instance, Encase forensic tool is used, and for Linux instance, SANS Investigative Forensic Toolkit (SIFT) is used. This processing generates web history, timeline construction and restores data on the disk snapshot for analysis. The post-processed data is later pushed to AWS S3 buckets. The State tracking database is updated after the completion

of this process; this database is used to update the customer regarding the status of the jobs being executed on acquired data to ensure transparency. Finally, the incident response team is notified about the process completion by using the AWS SNS service. For further analysis, forensic experts and security teams work on the processed data. Finally, the processed data is stored in an AWS storage account with a proper data retention policy. AWS applies legal hold policy [62] to preserve the data for investigation upon receiving a request from LEAs. This ensures that the data is retained with CSP until the policy is removed explicitly. Legal hold policy also ensures data integrity hence, making the evidence admissible.

A.2 Azure Workflow

The Azure monitor is used for collecting and analyzing data from Azure cloud resources. It is used to centralize data from cloud resources (i.e., user applications, operating system, Azure services, Azure subscription, etc.) to perform analytics and metrics [59]. To collect monitoring data from VM and other computing resources, it uses monitoring agents [114]. Based on the type of data required the Azure monitoring agent can be selected. More than one monitoring agent can be used based on the requirement. Azure monitoring agents are - Azure Monitor agent, Diagnostics extension, Log Analytics agent, Dependency agent. Log Analytics agent [115] is used to collect logs from Azure, other clouds, on-premise resources and store them in the log analytics workspace. Azure Monitor Logs feature of Azure Monitor is used to collect and analyze logs stored in the log analytics workspace. Further, Azure HDInsight is used to perform big data analytics on huge volumes of cloud logs [116].

We analyzed the forensic framework for Azure, based on Azure official documentation for forensics [53] [117]. The Security Operation Center (SOC) team is responsible for handling forensic procedures at Azure and ensures a valid chain of custody. The SOC team at Azure maintains different SOC subscriptions. Azure services with Role-Based Access Control (RBAC) to each subscription include the following services - 1. SOC VNet - this is a secure Azure virtual network. 2. SOC Automation Account - this is used for process automation and configuration services. It hosts Hybrid Runbook Worker VM; this provides all control mechanisms to capture the target VM snapshot. 3. SOC Storage Account - this is used to host Azure fileshare to compute the hash value of the target virtual instance and host copies of the target disk snapshots in immutable blob storage

in WORM state (i.e., Write Once and Read Many) to avoid data modification and deletion. 4. SOC Key vault - this is used to store snapshots' hash values and encryption keys to ensure integrity and confidentiality of disk content. 5. Log analytics workspace - this is used to store the activity logs to monitor all events on Azure SOC subscription to Azure Monitor.

In case of occurrence of an incident, the SOC team receives a request to capture digital evidence. SOC team member signs in to Azure SOC subscription and uses Hybrid Runbook Worker VM [52] in Azure Automation to trigger the forensic process. Hybrid Runbook Worker(HRW) is used to manage the resources on Azure, other clouds, or on-premise resources. It sits locally on on-premise resources or target VM and accesses the local resources. HRW runs Azure runbooks(Automated scripts) on target VM. In Azure forensics, HRW is exclusively used to execute CopyVMDigitalEvidence runbook on the target VM to collect the target VM disk snapshot. HRW is hosted on the target VM, in the same subnet that grants access to the SOC storage account using the service endpoint mechanism. HRW must have managed identity or service principal to access target VM's subscription to provide snapshot rights on target VM disks, access to SOC storage account, access policy for SOC key-vault to get and set secret keys. Hybrid runbook worker also ensures that Copy-VMDigitalEvidence runbook has all required permissions to access target virtual instance and SOC subscription.

Copy-VmDigitalEvidence runbook is responsible for collecting the snapshot from the target instance and storing it on the SOC storage account. Copy-VmDigitalEvidence runbook initially signs in to target virtual instance and SOC subscription. It creates a snapshot of the operating system and data disks, copies it to subscriptions immutable blob storage containers, and Azure temporary file share. Azure file share is used as a temporary repository to calculate the hash values of the snapshot and generate secret keys by using SHA-256, and AES-256 algorithms, respectively. Later, Copy-VmDigitalEvidence runbook copies the generated hash values and encryption keys to the SOC key vault and deletes all the copies of the snapshot from the temporary file share. The snapshot on immutable storage is used for further analysis by using Azure security services.

To provide access to evidence for investigators Storage Shared Access Signatures (SAS) URI is used to ensure granular control over access policy. Also, the investigators are provided with required encryption keys in the SOC key vault to decrypt the data for analysis. Alternatively, time-limited read-only SOC Storage account access to IP addresses from outside, on-premises networks are given to

investigators to download digital evidence. Further, Legal hold policy [63] is applied on the immutable storage containers for preserving the data and ensuring its integrity for legal proceedings.

A.3 GCP Workflow

Google Cloud's operations suite [118] is used to collect and monitor data from Google cloud resources. Google cloud logging [60] service in Google cloud operation suite is a fully managed service used to collect logs from the application, VM, Google cloud services. Google cloud logging service centralizes logs from all sources and performs advanced and custom filtering to identify suspicious activity. To collect logs from other clouds and on-premise resources Google cloud logging agent is installed on the target instance. Big Query is used for performing log analytics in the Google cloud. Further, the Google cloud monitoring [119] service in the Google cloud operation suite is used for performing analytics and monitoring the data collected by the Google logging service.

Forensics at GCP have been detailed in Google Cloud Next '18 annual event. A session by Sami Zuhuruddin, Solution Architect at Google, demonstrated the incident preparedness and forensic procedure implemented at GCP [55]. Forensics at Google include gathering forensic artifacts, identifying tools and processes, analysis pipeline, and process automation. The resources required to perform forensics on Google include - 1. Google Compute Engine (GCE) - this is the forensic virtual instance used to attach the disk snapshot for analysis. 2. Plaso - this is a framework used to retrieve timestamps from the source (disk snapshot) and arrange the events in chronological order to construct a super timeline. 3. Timesketch - this is a third-party tool for timeline analysis. This tool takes the Plaso file as input and organizes the timelines in a presentable manner. 4. Google Cloud Storage (GCS)- This is used to store the acquired disk snapshot for analysis and to preserve the evidence for the long term.

Create a new project under a secured environment for operational isolation, such that it enables one-way flow of data with restricted personnel access. Initially creates the snapshot of the target virtual instance and later a disk image from the snapshot. Calculate the hash value of the image file and encrypt the image file. Export the image file to GCS, and hash values, encryption keys to Google secret manager storage service. Restore the image file and use third-party forensic tools for analysis. Super timeline is created by using third-party tools like Plaso and

Timesketch. Super timeline details the occurrence of the events in chronological order; this helps in incident reconstruction.

Similar to the legal hold policy used by AWS and Azure, GCP uses Object hold [64] to retain the data. In case of legal investigation, it applies temporary object hold on the evidential data. Temporary object hold along with appropriate data retention policies ensures data integrity and preservation until the temporary object hold is removed exclusively by the CSP.

Google Cloud uses Google Rapid Response (GRR) [55] for memory forensics. GRR started as a project at Google and is now available as open-source at Github [120]. GRR is an agent-based application that can be installed on Windows, Macintosh, Linux. GRR has two main functionalities, Start flow and Hunt manager. Start flow is used to retrieve the memory dumps based on the target instance and process ID. Hunt manager is used for threat hunting in live suspicious environments.

GRR has a server agent and a client agent. The GRR server agent is installed on the security server of the Google Virtual Private Cloud (VPC), and the GRR client agent is deployed on the targeted virtual instance. The GRR client agent continuously polls the GRR server agent for work. We can trigger the memory dump process by sharing the process id of the suspicious process to get the memory dump of the associated process. Analysis of the acquired data is done by using third-party tools like Rekall, Volatility, etc.

Cloud Virtual Machine Forensics - An Anti-forensic Perspective

by Pranitha Sanda

Submission date: 27-May-2024 10:49AM (UTC+0530)
Submission ID: 2388999552
File name: 19MCPC01_Thesis-1.pdf (6.34M)
Word count: 36254
Character count: 184194

Cloud Virtual Machine Forensics - An Anti-forensic Perspective

ORIGINALITY REPORT

The overall plagiarism is 32% - 30% (25% + 4% + 1%) = 2%

32%	28%	31%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

Pranitha

Associate Professor
School of CIS
Prof. C.R. Rao Road
Central University
Hyderabad-46 (India)

PRIMARY SOURCES

1	link.springer.com Internet Source This is our own publication. Pranitha	25%
2	Pranitha Sanda, Digambar Pawar, V. Radha. "Blockchain-based tamper-proof and transparent investigation model for cloud VMs", The Journal of Supercomputing, 2022 Publication This is our own publication. Pranitha	4%
3	Pranitha Sanda, Digambar Pawar, V. Radha. "Chapter 47 VM Anti-forensics: Detecting File Wiping Using File System Journals", Springer Science and Business Media LLC, 2022 Publication This is our own publication. Pranitha	1%
4	Jan-Niclas Hilgert, Martin Lambertz, Shujian Yang. "Forensic analysis of multiple device BTRFS configurations using TheSleuth Kit", Digital Investigation, 2018 Publication	<1%
5	Pranitha Sanda, Digambar Pawar, V. Radha. "An insight into cloud forensic readiness by	<1%

	leading cloud service providers: a survey", Computing, 2022 Publication	
6	Submitted to University of Hyderabad, Hyderabad Student Paper	<1 %
7	Submitted to Colorado Technical University Student Paper	<1 %
8	Niranjan Reddy. "Practical Cyber Forensics", Springer Science and Business Media LLC, 2019 Publication	<1 %
9	www.mdpi.com Internet Source	<1 %
10	www.slideshare.net Internet Source	<1 %
11	core.ac.uk Internet Source	<1 %
12	ebin.pub Internet Source	<1 %
13	www.researchgate.net Internet Source	<1 %
14	blog.vidizmo.com Internet Source	<1 %
15	web.mit.edu Internet Source	

		<1%
16	www.coursehero.com Internet Source	<1%
17	www.geeksforgeeks.org Internet Source	<1%
18	Yuding Wang, Kacem Chehdi, Claude Cariou, Benoit Vozel. "Data Stream Unsupervised Partitioning Method", IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium, 2022 Publication	<1%
19	inst.eecs.berkeley.edu Internet Source	<1%

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		