electronics

MDPI

Article

Forensic Analysis of IoT File Systems for Linux-Compatible Platforms

Jino Lee [1] and Taeshik Shon [2,*]

[1] Department of Artificial Intelligence Convergence Network, Ajou University, Suwon 16499, Korea
[2] Department of Cybersecurity, Ajou University, Suwon 16499, Korea
* Correspondence: tsshon@ajou.ac.kr

Abstract: Due to recent developments in IT technology, various IoT devices have been developed for use in various environments, such as card smart TVs, and smartphones Communication between IoT devices has become possible. Various IoT devices are found in homes and in daily life, and IoT technologies are being combined with vehicles, power, and wearables, amongst others. Although the usage of IoT devices has increased, the level of security technology applied to IoT devices is still insufficient. There is sensitive information stored inside IoT devices, such as personal information and usage history, so if security accidents happen, such as data leakage, it can be very damaging for users. Since research on data storage and acquisition in IoT devices is very important, in this paper we conducted a security analysis, from a forensic perspective, on IoT platform file systems used in various environments. The analysis was conducted on two mechanical platforms: Tizen (VDFS) and Linux (JFFS2 and UBIFS). Through file system metadata analysis, file system type, size, list of files and folders, deleted file information were obtained so that we could analyze file system structure with the obtained information. We also used the obtained information to check the recoverability of deleted data to investigate the recovery plan. In this study, we explain the characteristics of platforms used in various environments, and the characteristics of data stored in each platform. By analyzing the security issues of data stored during platform communications, we aimed to help in solving the problems affecting devices. In addition, we explain the analysis method for file system forensics so that it can be referred to in other platform forensics.

Keywords: digital forensic; IoT devices; VDFS; JFFS2; UBIFS

Citation: Lee, J.; Shon, T. Forensic Analysis of IoT File Systems for Linux-Compatible Platforms. *Electronics* **2022**, *11*, 3219. https://doi.org/10.3390/electronics11193219

Academic Editor: Fernando De la Prieta

Received: 20 September 2022
Accepted: 4 October 2022
Published: 8 October 2022

1. Introduction

As the Internet of Things (IoT) advances and the market size increases, increasingly advanced technologies are being applied to various devices [1]. As of 2022, IoT technologies are being used in various devices and contexts, including vehicles, artificial intelligence, wearables, speakers, and industrial products, and it is becoming increasingly difficult to find devices without these technologies [2,3]. IoT devices store a variety of data acquired through interactions with networks, devices, and people. Such stored data includes sensitive user information, including system logs and personal information [4].

However, the level of security applied to IoT devices remains insufficient and devices such as speakers and intercoms are vulnerable to various attacks. In one case, an individual's IP camera was hacked, and the resulting video was sold to an Internet site [5]. IoT devices have become ubiquitous, and significant harm can occur through attacks. To obtain proper evidence, forensic and security technologies need to be developed [6–8].

As the fields in which IoT devices and technologies are applied continue to diversify, the numbers of IoT platforms are increasing. In addition, because each platform has a different data storage method and structure, knowledge of various platforms is required. Although research has been conducted on certain platforms, there are still platforms for which no studies have yet been conducted [9–12].

In addition, the types of IoT platforms and security technologies of the platforms are developing daily [13]. Due to the development of technologies, it is difficult to obtain data from IoT devices because of improved security and encryption. Therefore, research is needed on data acquisition and analysis methods in various environments to investigate and analyze devices.

Platforms are mounted on various IoT devices, so such devices can be targets for attack. IoT devices can store important information, such as user activity and usage records [14]. It is, therefore, necessary to research platforms to improve the security of IoT devices and the efficiency of evidence collection. With this in mind, we analyzed the metadata of file systems on various platforms for improved security and evidence collection. In addition, we propose a method to extract data through analyzed metadata and a method to recover deleted data. Finally, we investigate the types of artifacts that can be obtained for each file system by applying the proposed forensics method. The contributions of this study are as follows:

- Through a metadata analysis of the file systems mounted on IoT devices, information, such as the system type and size, list of files and folders, and deleted files, was obtained.
- The possibility of extracting files stored in a file system was analyzed experimentally. The metadata on devices using the Vertically Deliverable File System and Journaling File system version 2 (JFFS2) were analyzed to confirm the possibility of data extraction through file carving.
- The recoverability of deleted files in a file system was analyzed experimentally. Data on deleted files were confirmed using the snapshot and journaling functions applied by the file system.
- Analysis and file recovery experiments were conducted on real devices and it was confirmed that the research technique could be applied to other real IoT devices.

By analyzing the data storage structure and security issues of the platform, the security of IoT devices to be developed in the future can be improved. In addition, we present an approach for analyzing the platform mounted on IoT devices using file system forensics. The proposed approach allows for referencing when analyzing IoT device platforms that are newly developed or difficult to recognize.

The remainder of this paper is structured as follows. Section 2 describes existing studies related to file system forensics on various platforms. Section 3 details the characteristics of each platform and applicable devices, and Section 4 introduces the metadata analysis experiments conducted using a file system. Section 5 describes the extraction and recovery of files based on case studies. Section 6 discusses the results of such a study, and, finally, Section 7 provides some concluding remarks.

2. Related Work

IoT file system forensics requires both analysis of characteristics of IoT devices and knowledge of file systems. Since IoT has different characteristics for each device, a forensic framework that can combine various devices is needed.

In 2019, Jianwei Hou et al. confirmed the importance of forensics for IoT devices and investigated papers that conducted forensics for various IoT devices. Classifying the papers into forensic techniques, the study made it easier to refer to the necessary parts in future research [15].

In addition, other studies investigated various devices to which IoT technology can be applied and analyzed the problems facing IoT forensics. Through this, it was possible to identify devices to which IoT forensics can apply and to gauge the direction in which forensics technology should go [16,17].

In 2019, Arduino's research was conducted to simulate attacks that may occur on IoT devices, and to detect and identify attacks. Since this study simulated attacks that could easily occur in a network environment, the results of the study could help in the event of a real problem [18].

In 2022, the environment of IoT devices was formed through Raspberry Pie and forensics was conducted. This study, conducted on IoT, attempted to acquire data stored and communicated through connection with the network [19]. As such, various forensic studies were conducted on IoT devices, and could be applied to various devices.

A file system refers to a method of storing, maintaining, and managing data on a device. There are various types of file systems, and the method of managing data and the structure of storing data are all different for each file system. It is difficult to confirm the existence of a file system as a general user who uses only IoT devices. However, the file system contains a lot of information. As an example, there is information on deleted data from the stored data, which cannot be generally confirmed. To confirm this, metadata can be obtained through file system analysis, and various data can be accessed through the obtained information [20–22].

Since file systems are mounted on a wide variety of devices, there are various types, from file systems installed on basic computers and smartphones to file systems installed on small IoT devices [23,24]. As new types of devices are developed, the number of file systems suitable for devices is expected to continue to increase.

Research on file systems is being conducted, and file system forensics research is also being actively conducted in various fields, such as computers, smartphones, and IoT.

In 2019, a study was conducted to extract the files of a Samsung smart TV [25]. This study obtained and analyzed the data and extracted files through the chip-off technologies used by Samsung smart TVs and flash memory. However, the study only analyzed the file extraction and artifacts, and no research was conducted on Samsung's own file system.

In 2019, a forensic experiment was conducted on Amazon Echo [26]. The study applied a basic forensics model, and an analysis was conducted by dumping the firmware through UART communication. Based on a firmware analysis, various types of information, including the data type and MAC address stored in Amazon Echo, was obtained. However, this study only introduced an approach for a firmware analysis.

In 2019 and 2022, data analysis and extraction of data from various wearable devices were conducted [27,28]. This study considered the platforms of various wearable devices and organized the data stored on the platform, as well as the data acquisition. Although this study intensively analyzed the available evidence and file acquisition on various wearable device platforms, the file system layer was not considered.

In 2022, a study on smart home forensics using IoT was conducted [29], and the results indicated that data stored on a smart phone can be confirmed during communication with smart home IoT devices. However, only data stored on a smartphone, and not a smart-home IoT device, were analyzed.

In 2022, Android smartphones were analyzed in terms of the recovery and exportation of deleted data and a file recovery of the ext file systems [30]. In this study, forensics was conducted through metadata analysis of the file system used in Android smartphones.

In 2022, the application of IoT technology to vehicles, prompted forensic research on Android auto and apple CarPlay [31]. This study focused on the connection between devices and clouds, IVI devices and mobile devices, that is, network communication. However, it is significant in that it attempted to analyze new devices because analysis of IVI devices and mobile devices is necessary to obtain communication data stored in the device. In addition, research on data acquisition was conducted on Xiaomi Mi Smart Sensor, but not much on the file system stage [32]. It was a study on data recovery through the analysis of Google home and the analysis of SquashFS, a file system, and had a similar purpose to this paper. With the existence of forensic papers targeting various devices, it is possible to confirm the necessity of analysis at the file system stage of the IoT platform [33].

In addition to research on devices and platforms, framework research for forensic techniques has also been continuously studied [34–36]. However, since IoT devices are equipped with various platforms, further research on unresearched platforms is needed.

As such, research on IoT platforms is steadily progressing, but it is still insufficient. Some platforms have been analyzed at the file system stage, but there are still many

platforms that require further research. Related studies have focused on acquiring data from peripheral devices or analyzing files extracted from the device itself. However, it may be difficult to check deleted files by this method, and information on the acquired file may be limited.

To solve this problem, there is a need for a file system-based forensics technique that can obtain various data including deleted data. Although some file system forensics techniques have been studied, continuous research is required, because the characteristics of each platform or file system are different [37]. Therefore, in this study, a file system analysis was conducted to overcome these shortcomings.

3. IoT Platforms

In this section, we describe the characteristics of the platforms and file systems used by IoT devices. Various types of M2M equipment are currently being used, and so various platforms are also applied. In addition, the operating and file systems may differ, even for the same platform. Platforms applied to such devices include Fuchsia, Contiki, Tizen, and Linux.

Fuchsia is a platform developed by Google [38]. Fuchsia OS is currently experimentally installed in Google Nest Hub (1st Gen) and uses BlobFS and MinFS as the file systems. MinFS is a file system built for the Zircon kernel applied by Fuchsia. Fuchsia is expected to replace Android in the future.

Platforms are also used for low-power and low-capacity IoT devices. The Contiki platform is applied in low-power devices and uses Contiki OS and the coffee file system [39]. Contiki is run in memory, and because, it is mainly applied to sensors, lighting, and monitoring devices in smart cities, it is likely to be used in the future.

With the application of various platforms, to achieve improvements in forensic analysis, the necessity of conducting a platform analysis has been confirmed. Therefore, in this study, we conducted an experimental forensic analysis on two types of platforms: Tizen and Linux. The basic structures of the file systems used in Tizen and Linux was verified for this experiment.

3.1. Tizen (VDFS)

Tizen is a Linux-based platform developed by Linux, Samsung Electronics, and Intel, in 2012. Devices using the Tizen platform are primarily equipped with Tizen OS. Although various file systems can be used for Tizen OS, Samsung's own file system, VDFS, was applied during the experiment. VDFS is a Linux file system optimized for usage on eMMC flash devices. VDFS is a file system produced by Samsung that provides code as an open source by Samsung, but not much information is known about the file system. Through the analysis of the code provided by open source, it was found that VDFS used the btree structure when storing files, and deleted files were stored in the form of snapshots, rather than journal areas. Therefore, we could analyze the btree structure to extract stored files, and deleted files could also be extracted or recovered if snapshot data was not erased. Although VDFS is only used by Samsung, we conducted research on this file system owing to the high penetration rate of Samsung TVs. Figure 1 shows the system–layer structure of Tizen VDFS, which is configured in a manner similar to that of a basic Linux file system. VDFS operates on block devices, stores data in a block-based manner, and uses a btree structure for storage.

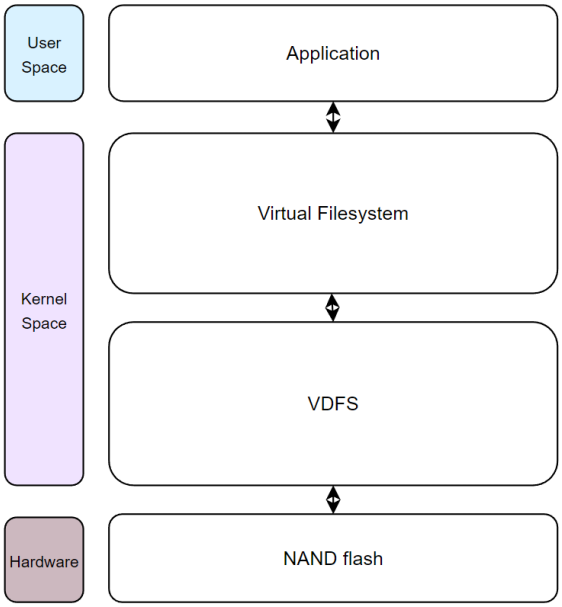

Figure 1. VDFS architecture on Tizen.

3.2. Linux (JFFS2 and UBIFS)

Linux is an open-source platform developed in 1991 and is mounted on various devices and PCs. Although the Linux OS used on the Linux platform can apply a variety of file systems, JFFS2 and UBIFS were used in the experiments. JFFS2 and UBIFS are Linux-based file systems used in flash memory devices. When storing files, JFFS2 can store files using three types of compression algorithms, zlib, rubin, and rtime, to manage data capacity more efficiently. Since JFFS2 is a journaling file system, files can be backed up and stored, and it is possible to recover the file even if the file is deleted later. UBIFS was developed based on improvements to JFFS2, and unlike JFFS2, it does not scan the entire media, and can, thus, be used for large capacity NAND flash memory. In addition, since JFFS2 stores file system indexes in memory, while UBIFS stores indexes in flash, there is a difference in performance and speed. When storing files, UBIFS encrypts file data through fcrypt, making it impossible to extract general files. When storing files, UBIFS encrypts file data through fscrypt, making it impossible to extract files in a conventional way. Therefore, with UBIFS using the journaling system, deleted files also have data, but it is difficult to recover deleted files due to fscrypt encryption. These file systems are mainly used in embedded systems, and although many years have passed since their initial development, they are still applied in certain models. Therefore, we conducted a metadata analysis and file extraction for the file system. Figure 2 shows the system–layer structures of JFFS2 and UBIFS. JFFS2 operates on memory technology device (MTD) devices, but UBIFS operates on UBI volumes and only on raw flash memory. For UBIFS, the UBI volume is used to access the MTD devices. In addition, UBIFS store data using a logical erasable block (LEB).

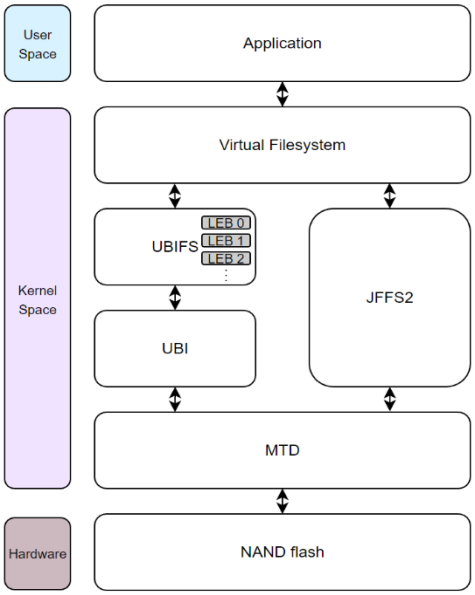

Figure 2. JFFS2 and UBIFS architecture on Linux.

4. Experiments and Analysis

In this section, experiments conducted to acquire data from a device are described, along with an analysis on the platform and file system used.

4.1. Experiments on Tizen Using VDFS

For a VDFS analysis of the Tizen platform, a study was conducted on a smart TV (model KU43T5300AFXKR). An analysis of the PCB of the model, as shown in Figure 3, revealed that a THGBMNG5D1LBAIT NAND flash chip with 4 GB of memory and 153 BGA was used. The chip-off technique was applied to obtain data for the corresponding NAND flash. Raw data was obtained through a DS3000-USB3.0-emmc153 reader, a product of Allsocket in Dongguan, China. Based on an analysis of the raw data obtained through the FTKimager analysis tool (v4.3.0.18), 26 partitions were recognized, as shown in Figure 4. Among the 26 partitions, 5 were confirmed to be VDFS partitions with VDFS 2007 magic numbers. A file system analysis was conducted only on the VDFS partitions.

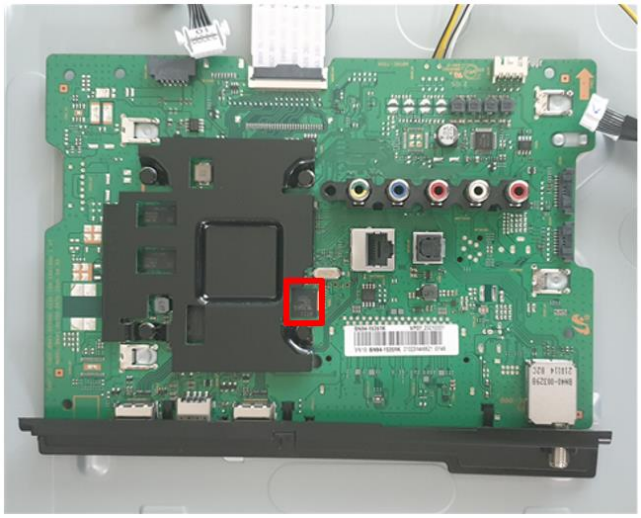

Figure 3. Nand flash chip (KU43T5300AFXKR) present on pcb of smart TV.

Figure 4. Smart TV partition.

For the analysis of the VDFS file system, we analyzed the code of the VDFS released by Samsung and proceeded. VDFS, a file system manufactured by Samsung, has a magic number labeled VDFS2007 at the front address of the partition, indicating that the partition was a VDFS. The superblock could be checked at the 0x400 location of the VDFS partition. The structure of the superblock is shown in Figure 5, and basic information on the partition, such as the version, creation time, and volume name, could be confirmed. Behind the superblock was an extended superblock with additional information at the 0x600 location. The structure of the extended superblock is as shown in Figure 6, and information on the stored data, such as the number of files, folders, and meta tree offset in the volume, could be found.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	signature				version				maximum block count							
10	creation timestamp															
20	creation timestamp												volume uuid			
30	volume uuid												volume name			
40	volume name												mkfs version			
50	mkfs version															
...	...															
A0	...				log blk size	...										

Figure 5. VDFS superblock structure in form of hex value.

An analysis of a superblock allowed us to obtain the offset of the meta tree, and when we moved to the acquired address, the xattr tree, catalog tree, inode bitmap, and fsm bitmap were located. The tree haf a magic number of 0x644E at the beginning of the block, and bitmap had magic numbers of 'inob' and 'fsmb'. In addition, the xattr tree had 'XAre' as its magic number. The xattr tree stored records such as the extension properties and access control, rather than regular files, and the catalog tree stored records of the files and folders. We could check the file and folder names in the catalog tree. Figure 7 shows a record of the airplay-service.md file in the catalog tree.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	file count								folder count							
10	volume start								volume end							
20	mount counter				sync counter				umount counter				generation inode number generation			
30	debug area															
40	btree extents total block count				padding				tables							
50	tables								meta btree start offset							
60	meta btree length															

Figure 6. VDFS extended superblock structure in form of hex value.

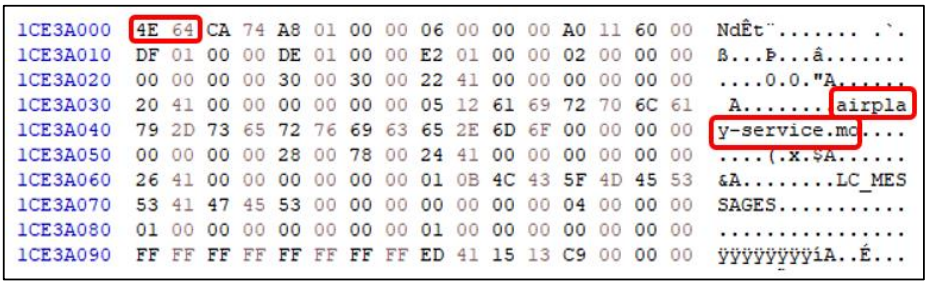

Figure 7. The airplay-service.md file records in a catalog tree with magic number 0x644E.

The catalog tree contained all the information on the files and folders. Figure 8 shows the structure of a file in the catalog tree. The type of data could be checked through the variable value 'type'. A value of 'type' 0x01 indicated a folder; 0x02 was a file; and 0x05 was a link record. Each dataset consisted of a key + record, which contained basic information, such as the name, type, and object ID. The record contained the size, creation/modification/access time, and offset address information of the file where the data were stored. However, if the type was a folder, the number of files present in the folder was written instead of the file size. If the type was a link record, only a key value was present.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	key len		record len		object id								parent id			
10	parent id				type	File name len	file name						-			key len end
20	-				file size (byte)								-			
30	-															
40	-								creation time							
50	creation time				modification time											
60	access time												file size (byte)			
70	file size (byte)				number of all blocks								data offset			
80	data offset				number of blocks_1								start block			
90	start block				data offset_2								number of blocks_2			
A0	number of blocks_2				start block_2								...			

Figure 8. Catalog tree data structure in form of hex value.

VDFS used snapshots to store the file system formation. Snapshots store data in files and folder-like images. Figure 9 shows the 'browser-data.db' file, and we confirmed that the offset addresses of the stored data differed because the contents changed as the 'browser-data.db' file was modified, and the previous files were saved as snapshots. If we looked at the offset corresponding to each file, data in the 'browser-data.db' file were present at that time. In this manner, we could extract or recover files using the snapshot function. Therefore, experiments related to file extraction and recovery were conducted to analyze this possibility, and the effects were verified through a case study.

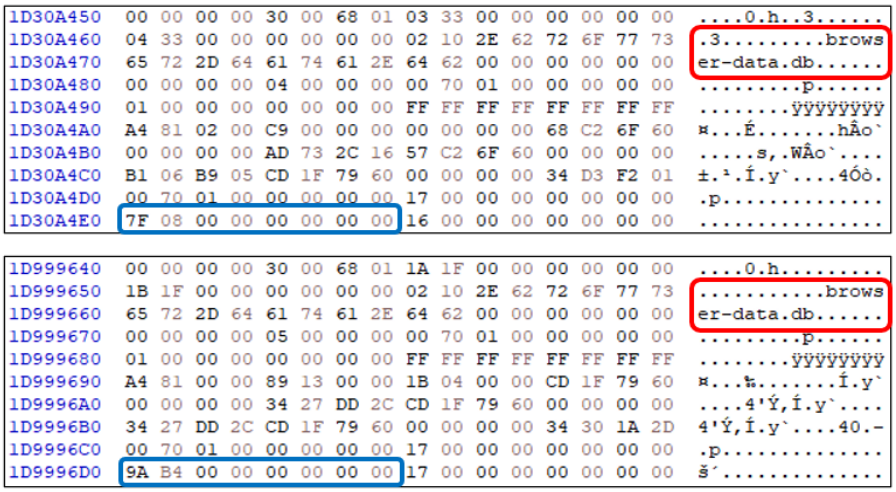

Figure 9. Two different offsets (**blue**) of 'browser-data.db' file (**red**).

4.2. Experiments on Linux Using UBIFS/JFFS2

In this study, we investigated the data central units (DCUs) of LOENK, a power device product. As shown in Figure 10, 256 MB NAND flash with a K9F2G08UOC chip was used. To obtain data from NAND flash, serial communication was executed through the RS-232 port of the PCB. As a result of accessing and checking the root account through serial communication, three MTDs were identified. After creating an SSH server for the extraction of the MTD, we dumped the MTD using a scp command to transfer files through SSH on the device. As a result of analyzing the MTD, it was confirmed that a kernel, JFFS2, and UBIFS were installed. The file systems were also analyzed.

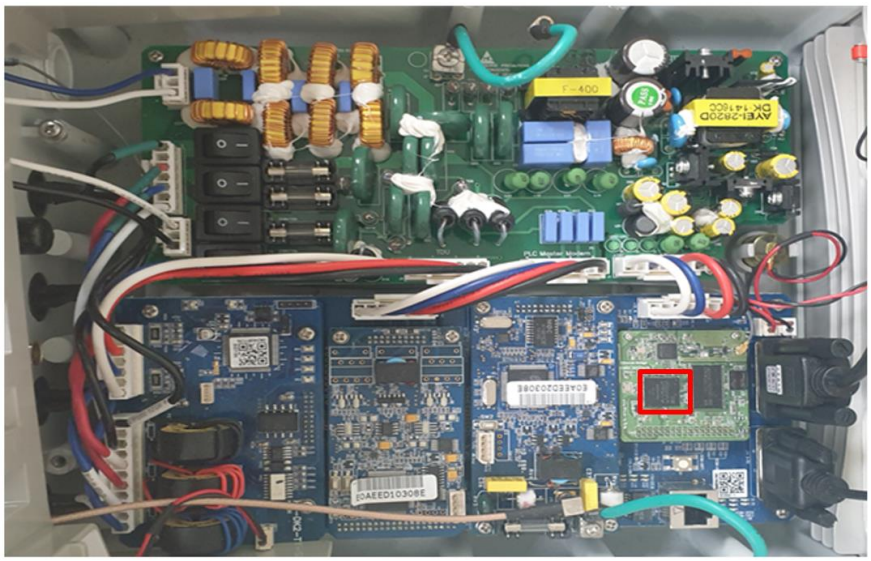

Figure 10. PCB of DCU.

For the analysis of JFFS2 and UBIFS, an analysis of the code disclosed as an open source was conducted. JFFS2 had a magic number labeled 0x1985 at the front of the node, and the unused area was filled with 0xFF values. JFFS2 consisted of several node types. Among the nodes, Figure 11 shows the structure of the inode, which had a node type of 0xE002. In addition, the inode contained various information on the file, including file size, modification, access time, and data. JFFS2 compressed and stored data for efficient data management, and only zlib, rubin, rtime, and LZO were used for the compression algorithm.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	0x1985		node type		node_len				header_crc				inode number			
10	version				file mode				uid		gid		isize			
20	atime				mtime				ctime				Offset			
30	compressed size				decompressed size				comp	usr comp	flags		data_crc			
40	node_crc				data											

Figure 11. JFFS2 inode structure in form of hex value.

In JFFS2, a dirent node indicated the inode. Figure 12 shows the structure of the dirent node, and the corresponding node showed the file name and modified time stored in the inode. Based on the inode number of the corresponding node, an inode had the same inode number value, and the location where data existed could, thus, be determined.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	0x1985		node type		node_len				header_crc				parent inode			
10	version				inode number				mctime				name size	type	unused	
20	node_crc				name_crc				name							

Figure 12. JFFS2 dirent node in form of hex value.

Next, UBIFS was analyzed. UBIFS operates through the UBI volume, sp the UBIFS region had to be extracted from this volume if data from the MTD were to be obtained. In the UBI, an EC header containing information on the volume for each block existed, and in the EC header, basic information, such as the number of errors and the start offset of the data, was stored. In addition, we could move to the vid header through the vid header offset of the EC header. The vid header stored information on volumes that did not exist in the EC header, such as the LEB size and number of LEBs used in the volume.

UBIFS image extraction was possible using Github ubi reader (https://github.com/jrspruitt/ubi_reader, accessed on 5 May 2022). In this study, UBIFS images were extracted using ubi reader v0.8.0 in a Linux environment. UBIFS was in the form of a node. Figure 13 shows the structure of a header, which is a common form of all nodes, and all nodes in UBIFS had 0x06101831 as their magic number placed in front.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	magic number				crc				sqnum							
10	len				node type	group type	padding		-							

Figure 13. UBI common header in form of hex value.

The first node in UBIFS was a superblock. In a superblock node, basic information, such as the LEB size and LEB number of UBIFS, could be found; in addition, the LEB size became important when analyzing the metadata.

The second LEB had a master node. Figure 14 shows the structure of the master node, which was used in a similar manner as the superblock node. Information on the volume might also be stored, such as the size of the available area and the number of LEBs used. The master node also contained information regarding the location of the index node of the root and we could use this information to find the index node.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	-								highest_inum							
10	cmt_no								flags				log_lnum			
20	root_lnum				root_offs				root_len				gc_lnum			
30	ihead_lnum				ihead_offs				Index_size							
40	total_free								total_dirty							
50	total_used								total_dead							
60	total_dark								lpt_lnum				lpt_offs			
70	nhead_lnum				nhead_offs				ltab_lnum				ltab_offs			
80	lsave_lnum				lsave_offs				lscan_lnum				empty_lebs			
90	ldx_lebs				leb_cnt				padding							

Figure 14. UBIFS master node in form of hex value.

Index nodes had information on the LEB numbers and node offsets, and when moving along the offset pointed to by a node, nodes with various information regarding the files, including the inode, data nodes, and directory entry nodes, could be accessed. Directory nodes had inode numbers for the files and folders, and inode and data nodes had data on the files. The structure of a data node, which stored data on the file, is shown in Figure 15.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	-								key							
10	key								size				compr_ type		compr_ size	
20	data															

Figure 15. UBIFS data node in form of hex value.

5. Case Studies

In this section, we analyze a file system based on a case study. The case study proceeded with the file extraction and recovery of deleted files using the file system. The file extraction proceeded with file carving by searching for file data through a metadata analysis. The recovery of deleted files proceeded by comparing such files with areas where data existed.

5.1. File Extraction

5.1.1. VDFS

To extract data from VDFS, a catalog tree had to be found. To access the catalog tree, we had to find the meta tree start offset in the extended superblock. The xattr tree, catalog tree, inode bitmap, and fsm bitmap were located in this offset. We could find a catalog tree to check the file record. Figure 16 shows the record for the 8669.png file, as well as information on the file name, file modification/generation time, and offset where the file data were stored. Information was also available on the total number of blocks where the data were stored, as well as the number of blocks stored in each offset.

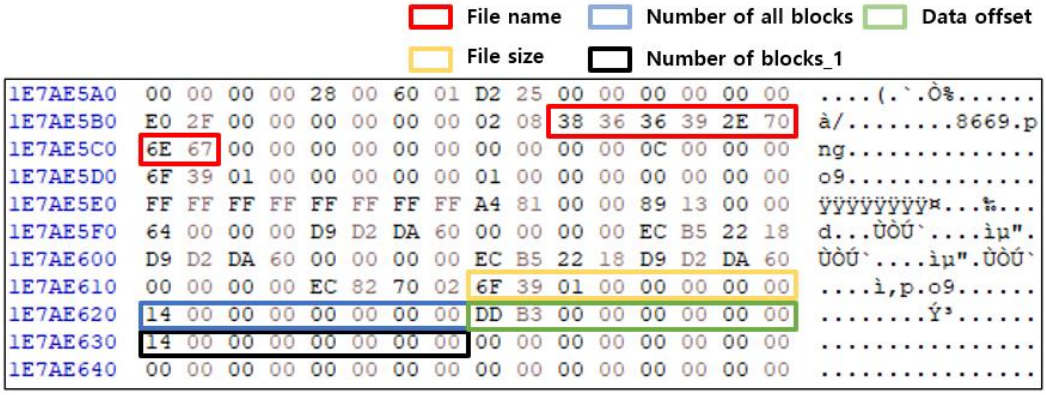

Figure 16. File record representing metadata where the 8669.png file is stored.

Therefore, the offset in which the data were stored might be known by multiplying the value (0xB3DD) representing the data offset based on the block size. When moving to the offset, it could be confirmed that data on the 8669.png file existed, as shown in Figure 17. Thereafter, a file could be extracted by performing data carving on an area as large as the data size.

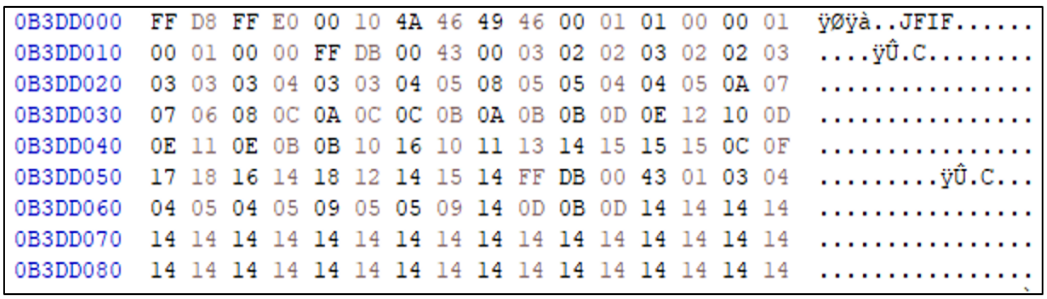

Figure 17. 8669.png file data.

5.1.2. JFFS2

To extract a file from JFFS2, it was necessary to determine the inode of the file. However, because file name data did not exist in the inode, we needed to find the dirent node to check the file name. A file might be extracted by finding an inode with the same inode number as the rent node. However, because JFFS2 compressed files when they were stored, file extraction required decompression. Figure 18 shows the dirent node and inode of test3.txt. The file name was located in the dirent node, and data could be found in the inode.

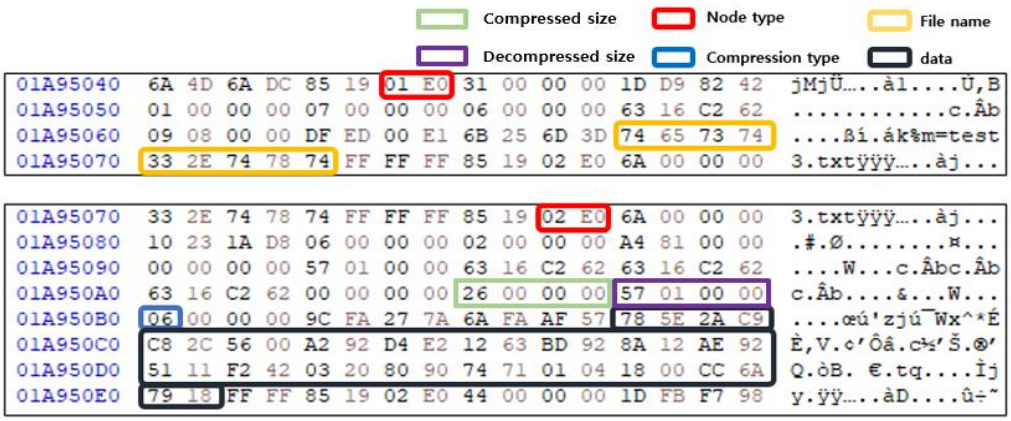

Figure 18. Dirent node (**top**) and inode (**bottom**) of 'test3.txt' file.

Although data were present in the inode, they were compressed, and the compression type could be checked. The compression type was mainly 0x06, which indicated that the data had been compressed using zlib. In order to confirm the contents of the compressed data, decompression should be performed in the same manner as the compressed type. The data was compressed into zlib, and in the case of zlib, it could be decompressed through a simple python code. Figure 19 is a code for decompressing a file compressed with zlib. The data size of the compressed file could be found through the compressed size metadata, and the original data could be obtained through the code of Figure 19 after carving the data area. The size of the decompressed file could be compared with the decompressed size metadata to verify that the file had been decompressed properly.

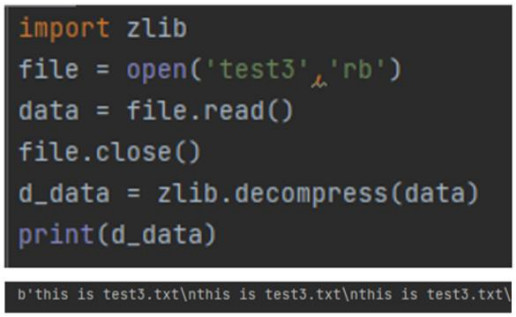

Figure 19. Zlib Python code to decompress files and the result of decompression.

5.1.3. UBIFS

A data node had to be accessed to extract the UBIFS files. To do this, we needed to move to the master node, which was placed in the second LEB. The indexing node offset was obtained and we moved through an analysis of the master node. We could then follow the branch of the indexing node to observe the directory entry and inode of the files and folders in that directory. Figure 20 shows the directory entry and inode, which can be distinguished by the node type value.

By searching for the inode number of the file, the file data node could be found. In this case, the last value of the inode number of the data node was changed to 0x20. Figure 21 shows the directory entry and data node of the k2logd.sh file. By comparing the inode number values, we could see that it was the same file.

However, unlike JFFS2, UBIFS applied compression and encryption in the data area, which made it difficult to extract the files. Figure 22 shows the data of the dcu.tar file and the data node of this file stored in the UBIFS. It can be seen that these data differed from existing data owing to the encryption and compression of the UBIFS. Thus, in the case of a UBIFS file, decoding was required for extraction.

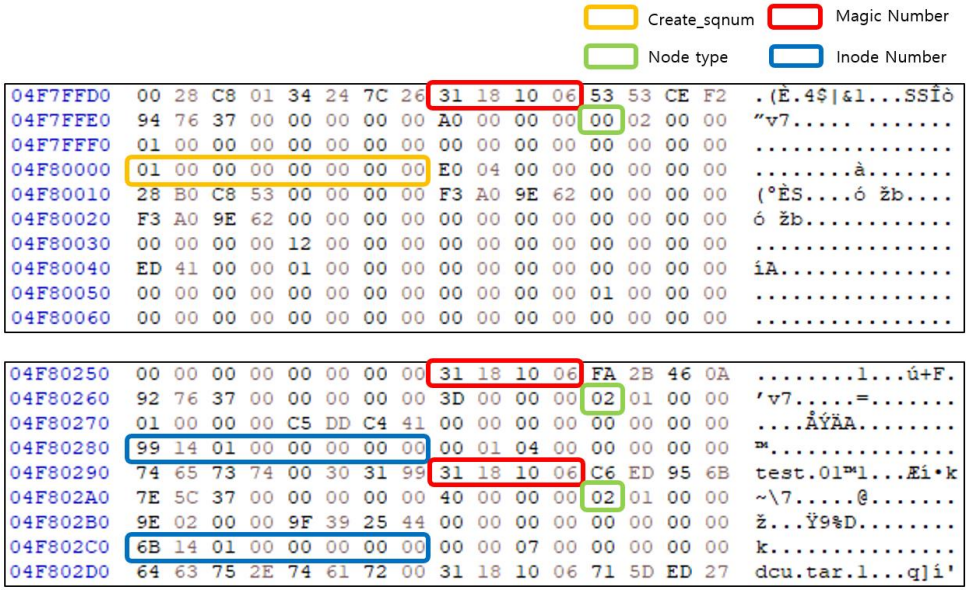

Figure 20. UBIFS directory of entry (**top**)/inode (**bottom**).

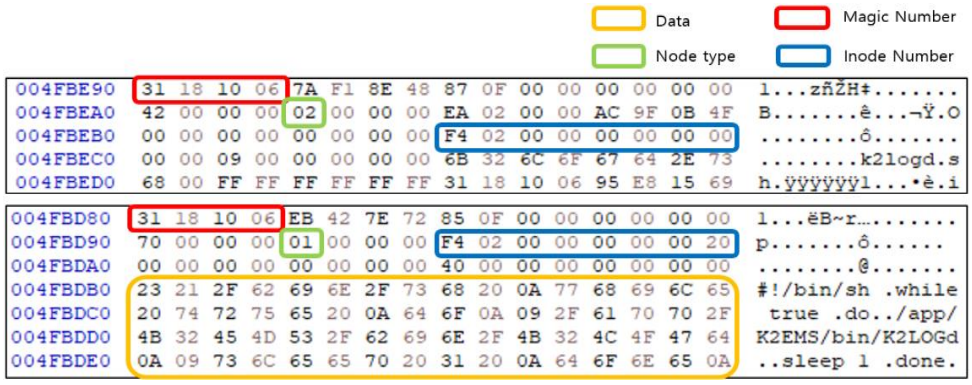

Figure 21. UBIFS directory of entry (**top**)/data node (**bottom**).

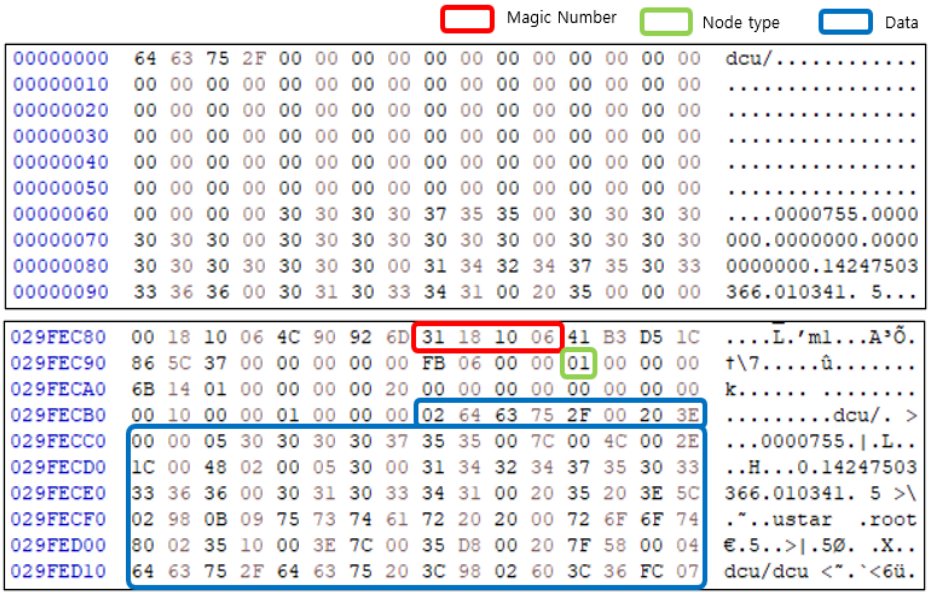

Figure 22. Actual file data in dcu.tar file(**top**)/data node which is dcu.tar file is stored (**bottom**).

5.2. File Recovery

5.2.1. VDFS

In the case of VDFS, data were stored in the form of snapshots, enabling the recovery of the deleted files. We could search for the deleted file name in the catalog tree or check the file size and records that did not have an offset address where the data were stored. Figure 23 shows the deleted 8669.png file's record. We confirmed that the file size and data area offset address did not exist. This phenomenon occurs when a file is either modified or deleted. However, because VDFS used a snapshot feature, if a new snapshot was not overwritten after the file was deleted, information on the deleted file existed in the previously saved snapshot. Therefore, it was possible to recover deleted data using the previous snapshot information.

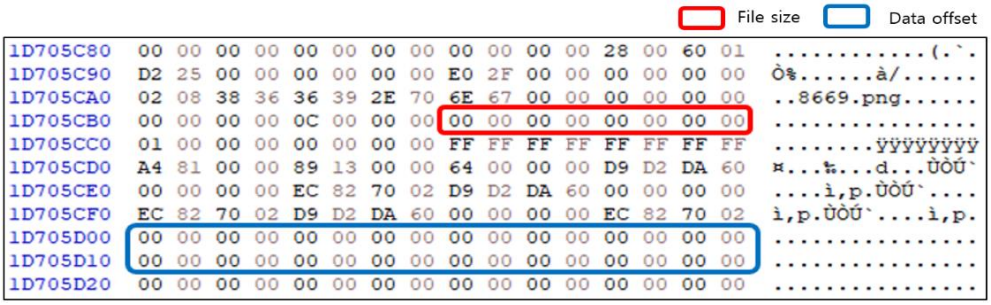

Figure 23. Record of deleted 8669.png file without file size and offset.

Using an editor such as HxD, we could obtain deleted file information from previous snapshots by searching for the file name, or for the file name that we wanted to recover. In the previous saved snapshots, an offset value containing the size and data of the 8669.png file existed, as shown in Figure 24. It was, therefore, possible to recover the deleted file by moving it to the corresponding offset address and copying and storing the data in the corresponding area.

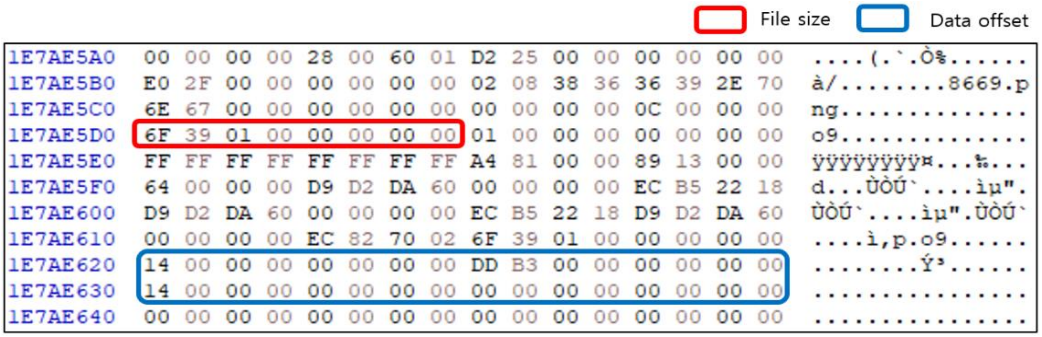

Figure 24. Records of the 8669.png file stored in snapshot format.

5.2.2. JFFS2

JFFS2 is a journaling file system, so it was also possible to recover deleted files. After deleting the test2.txt file, a partition dump was applied in the present study. Figure 25 shows the inode and dirent node of the test2.txt file. The left side of Figure 25 shows that the version was 0x04 with data prior to the deletion of the file, and the right side shows that the version changed to 0x05 with data after the file was deleted. Therefore, when analyzing the JFFS2 file system, if a deleted file with only a dirent node was found, a search for the inode prior to the file deletion could be conducted, and the deleted file could be recovered.

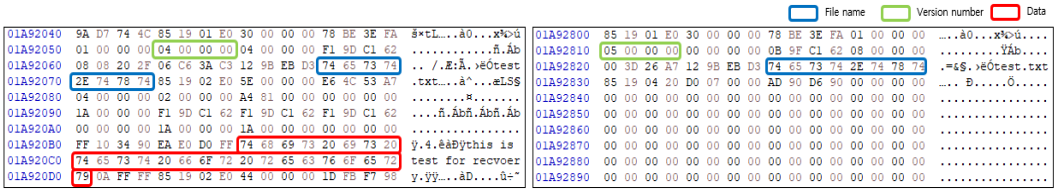

Figure 25. Recovery of deleted files in JFFS2 by verifying file versions.

5.2.3. UBIFS

UBIFS applied compression and encryption when files were stored. As shown in Figure 26, data of dcu.tar file had been encrypted and compressed. Due to this phenomenon, recovery of deleted files in UBIFS was limited.

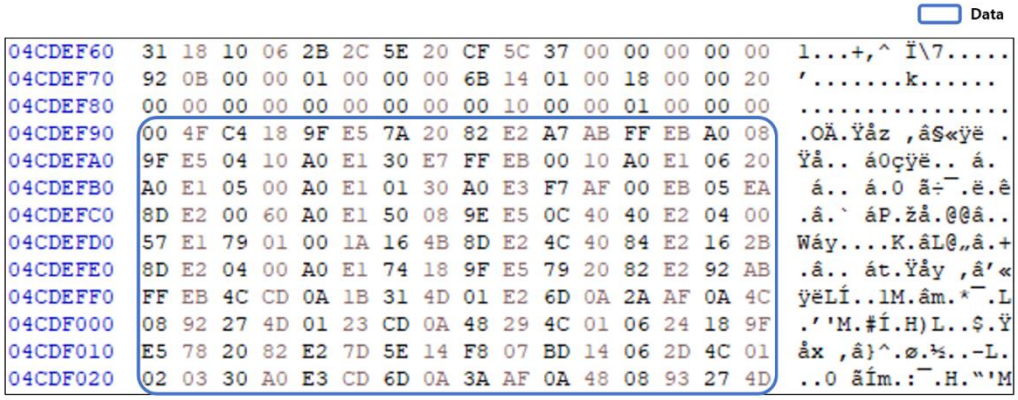

Figure 26. Encrypted data area in dcu.tar file.

Through this experiment, the possibility of extracting and recovering files of three types of file systems was studied. According to the results of the study, the possibility of checking the file name by file system, the possibility of file extraction, and the possibility of recovering deleted files were confirmed, which can be seen in Table 1.

Table 1. The possibility of information available for each file system.

File System	File Name Check	File Extract	File Recovery
VDFS	Able	Able	Able
JFFS2	Able	Able	Able
UBIFS	Able	Disable	Disable

6. Discussion

We conducted experiments on file system analysis, file extraction, and the possibility of recovery of deleted files on IoT devices. Since the experiment used a real device in the experiment, the applied analysis method could also be used in other models.

In this study, file system forensics was conducted for Samsung smart TV. As a result of analyzing Samsung smart TV, it was confirmed that Samsung's own file system, called VDFS, was used. By analyzing metadata, we understood the structure in which the file was stored and the metadata information. Access to the file data area was required to verify file extraction and recoverability of the file system, which was accessible through the previously performed metadata analysis. Studies have shown that VDFS stores data in the form of snapshots, and past snapshots allowed extraction of files and recovery of deleted files.

In addition, file system forensics was conducted on dcu, a power device, as an additional device. As a result of analyzing dcu, it was confirmed that two file systems, JFFS2 and UBFIS, were used. The file systems were also analyzed to confirm metadata and file

structures and research on the possibility of file extraction and file recovery. JFFS2 uses a journaling system to extract and recover deleted files. However, since the file is compressed and stored, extraction of the file requires decompression, and decompression is possible through a simple python code. UBIFS is also compressed, but since the file is encrypted, the contents of the file cannot be checked even if decompression is performed.

The results of this study can help in the extraction and recovery of files for IoT devices using the same platform. In addition, if only device data can be obtained, the technology used in this study has the advantage of being able to apply forensics quickly anywhere and does not require additional cost or technology. In particular, this paper's research was conducted at the file system stage on VDFS, which was previously conducted only with file recovery research. In addition, we analyzed metadata of some file systems installed in IoTs, which have not been implemented much in the past, and this differentiated our study from studies that focused only on file recovery.

However, some limitations existed in this study. In the case of UBFIS, compression and encryption proceed when the file is saved, but the method for decryption could not be confirmed. Accordingly, there was a limitation in that original data stored in UBIFS could not be collected. It also required analysis of some metadata. Analysis did not proceed because it was difficult to confirm what information some metadata stored, which made it difficult to find the desired file in the directory. For example, additional analysis of metadata, such as the data table, required finding the desired file. Therefore, in a future study, additional metadata will be analyzed to enable faster file navigation. In addition, for the file extraction of UBIFS, research on the decryption of encrypted data areas is needed. In addition, further research will be conducted on platforms and file systems used in small IoT devices.

7. Conclusions

Various IoT platforms are currently in use, and analysis studies applying to IoT platforms are extremely important from a forensic or security maintenance perspective. As various platforms are used, various evidence acquisition approaches are needed in the event of a crime. In this study, we analyzed the file systems VDFS, JFFS2 and UBIFS which are used on the Tizen and Linux platforms. This study presented an approach for conducting a metadata analysis, as well as an approach for data acquisition and recovery when analyzing other file systems for reference. In this study, the file storage structure of a specific file system was analyzed through metadata analysis. It also showed the possibility of extracting stored files by suggesting methods, such as decompression and decryption, according to methods, such as encryption and compression, used when storing files. In addition, when the file was deleted, a method of recovering the deleted file was studied according to the method of storing the file in the file system, such as journaling and snapshots. Finally, the division into file name verification, file extraction, and file recovery stages facilitates checking the applicability of the file system step by step. Through this, more diverse data can be secured from IoT devices and used as evidence in the event of a crime. Moreover, based on the analysis results of this study, future IoT devices with greater security can be developed. Through the metadata analysis, it is expected that the results of this study will contribute to securing evidence in the event of a crime. In particular, since VDFS analysis has not been previously studied, it may be helpful to secure evidence from smart TVs through this study. In addition, it is expected to help secure evidence when investigating IoT devices using file systems with similar structures. Therefore, the metadata analysis on the file system conducted in this study is expected to help improve the overall security technology of the IoT platform, as well as the forensics aspect.

Author Contributions: Conceptualization, J.L. and T.S.; methodology, J.L. and T.S.; validation, J.L.; formal analysis, J.L.; investigation, J.L.; writing—original draft preparation J.L.; writing—review and editing, J.L. and T.S.; project administration, J.L.; All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Korea Electric Power Corporation.(Grant number: R21XO01-45).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cui, J.; Cui, L.; Huang, Z.; Li, X.; Han, F. IoT Wheelchair Control System Based on Multi-Mode Sensing and Human-Machine Interaction. *Micromachines* **2022**, *13*, 1108. [CrossRef] [PubMed]
2. Jo, W.; Kim, S.; Kim, H.; Shin, Y.; Shon, T. Automatic whitelist generation system for ethernet based in-vehicle network. *Comput. Ind.* **2022**, *142*, 103735. [CrossRef]
3. Kim, S.; Jo, W.; Shon, T. APAD: Autoencoder-based payload anomaly detection for industrial IoE. *Appl. Soft Comput.* **2020**, *88*, 106017. [CrossRef]
4. Zhou, H.; Deng, L.; Xu, W.; Yu, W.; Dehlinger, J.; Chakraborty, S. Towards Internet of Things (IoT) Forensics Analysis on Intelligent Robot Vacuum Systems. In Proceedings of the 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), Las Vegas, NV, USA, 25–27 May 2022; IEEE: New York, NY, USA, 2022.
5. Hackers Gain Access to Home Security Camera Footages, Share Over 3TB Worth Videos. Available online: https://www.ibtimes.sg/hackers-gain-access-home-security-camera-footages-share-over-3tb-worth-videos-online-52439 (accessed on 21 July 2022).
6. Sungmoon, K.; Yoo, H.; Shon, T. IEEE 1815.1-based power system security with bidirectional RNN-based network anomalous attack detection for cyber-physical system. *IEEE Access* **2020**, *8*, 77572–77586.
7. Mehran, P.; Ekbatanifard, G. An efficient forensics architecture in software-defined networking-IoT using blockchain technology. *IEEE Access* **2019**, *7*, 99573–99588.
8. Dhelim, S.; Ning, H.; Farha, F.; Chen, L.; Atzori, L.; Daneshmand, M. IoT-enabled social relationships meet artificial social intelligence. *IEEE Internet Things J.* **2021**, *8*, 17817–17828. [CrossRef]
9. Kim, M.; Shin, Y.; Jo, W.; Shon, T. Digital forensic analysis of intelligent and smart IoT devices. *J. Supercomput.* **2022**. [CrossRef]
10. Shin, Y.; Kim, H.; Kim, S.; Yoo, D.; Jo, W.; Shon, T. Certificate injection-based encrypted traffic forensics in AI speaker ecosystem. *Forensic Sci. Int. Digit. Investig.* **2020**, *33*, 301010. [CrossRef]
11. Jo, W.; Shin, Y.; Kim, H.; Yoo, D.; Kim, D.; Kang, C.; Jin, J.; Oh, J.; Na, B.; Shon, T. Digital forensic practices and methodologies for AI speaker ecosystems. *Digit. Investig.* **2019**, *29*, S80–S93. [CrossRef]
12. Vilches, V.M.; Kirschgens, L.A.; Gil-Uriarte, E.; Hernández, A.; Dieber, B. Volatile memory forensics for the robot operating system. *arXiv* **2018**, arXiv:1812.09492.
13. Jonas, P.; Dewald, A. Forensic apfs file recovery. In Proceedings of the 13th International Conference on Availability, Reliability and Security, Hamburg, Germany, 27–30 August 2018.
14. Bharadwaj, N.K.; Singh, U. Acquisition and analysis of forensic artifacts from raspberry pi an internet of things prototype platform. In *Recent Findings in Intelligent Computing Techniques*; Springer: Singapore, 2019; pp. 311–322.
15. Hou, J.; Li, Y.; Yu, J.; Shi, W. A survey on digital forensics in Internet of Things. *IEEE Internet Things J.* **2019**, *7*, 1–15. [CrossRef]
16. Alex, A.-B.; John, A.; Mukherjee, T. Iot Software & Hardware Architecture and Their Impacts On Forensic Investigations: Current Approaches And Challenges. *J. Digit. Forensics Secur. Law JDFSL* **2021**, *16*, 1–19.
17. Hyunji, C.; Choo, K.-K.R. The need for Internet of Things digital forensic black-boxes. *Wiley Interdiscip. Rev. Forensic Sci.* **2020**, *2*, e1385.
18. Randi, R.; Hikmatyar, M. Investigation Internet of Things (IoT) Device using Integrated Digital Forensics Investigation Framework (IDFIF). *J. Phys. Conf. Ser.* **2019**, *1179*, 012140.
19. Mazhar, M.S.; Saleem, Y.; Almogren, A.; Arshad, J.; Jaffery, M.H.; Rehman, A.U.; Shafiq, M.; Hamam, H. Forensic Analysis on Internet of Things (IoT) Device Using Machine-to-Machine (M2M) Framework. *Electronics* **2022**, *11*, 1126. [CrossRef]
20. Seokjun, L.; Shon, T. Improved deleted file recovery technique for Ext2/3 filesystem. *J. Supercomput.* **2014**, *70*, 20–30.
21. Kim, H.; Kim, S.; Shin, Y.; Jo, W.; Lee, S.; Shon, T. Ext4 and XFS File System Forensic Framework Based on TSK. *Electronics* **2021**, *10*, 2310. [CrossRef]
22. Lee, H.; Chung, T. A Virtual File System for IoT Service Platform Based on Linux FUSE. *IEMEK J. Embed. Syst. Appl.* **2015**, *10*, 139–150. [CrossRef]
23. Zhang, K.; En, C.; Qinquan, G. Analysis and implementation of NTFS file system based on computer forensics. In Proceedings of the 2010 Second International Workshop on Education Technology and Computer Science, Wuhan, China, 6–7 March 2010; IEEE: New York, NY, USA, 2010.
24. Ohad, R.; Bacik, J.; Mason, C. BTRFS: The Linux B-tree filesystem. *ACM Trans. Storage (TOS)* **2013**, *9*, 1–32.
25. Nemayire, T.; Ogbole, A.; Park, S.; Kim, K.; Jeong, Y.; Jang, Y. A 2018 Samsung Smart TV Data Acquisition Method Analysis. *J. Digit. Forensics* **2019**, *13*, 205–218.
26. Li, S.; Choo, K.K.R.; Sun, Q.; Buchanan, W.J.; Cao, J. IoT forensics: Amazon echo as a use case. *IEEE Internet Things J.* **2019**, *6*, 6487–6497. [CrossRef]
27. Kim, S.; Jo, W.; Lee, J.; Shon, T. AI-enabled device digital forensics for smart cities. *J. Supercomput.* **2022**, *78*, 3029–3044. [CrossRef]

28. MacDermott, Á.; Lea, S.; Iqbal, F.; Idowu, I.; Shah, B. Forensic analysis of wearable devices: Fitbit, Garmin and HETP Watches. In Proceedings of the 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Canary Island, Spain, 24–26 June 2019; IEEE: New York, NY, USA, 2019.
29. Shinelle, H.; Karabiyik, U. Forensic Analysis of the August Smart Device Ecosystem. In Proceedings of the 2020 International Symposium on Networks, Computers and Communications (ISNCC), Montreal, QC, Canada, 16–18 June 2020; IEEE: New York, NY, USA, 2020.
30. Kim, H.; Shin, Y.; Kim, S.; Jo, W.; Kim, M.; Shon, T. Digital Forensic Analysis to Improve User Privacy on Android. *Sensors* **2022**, *22*, 3971. [CrossRef]
31. Shin, Y.; Kim, S.; Jo, W.; Shon, T. Digital Forensic Case Studies for In-Vehicle Infotainment Systems Using Android Auto and Apple CarPlay. *Sensors* **2022**, *22*, 7196. [CrossRef]
32. Gómez, J.M.C.; Carrillo-Mondéjar, J.; Martínez, J.L.M.; García, J.N. Forensic analysis of the Xiaomi Mi Smart Sensor Set. *Forensic Sci. Int. Digit. Investig.* **2022**, *42*, 301451. [CrossRef]
33. Barral, H.; Jaloyan, G.A.; Thomas-Brans, F.; Regnery, M.; Géraud-Stewart, R.; Heckmann, T.; Souvignet, T.; Naccache, D. A forensic analysis of the Google Home: Repairing compressed data without error correction. *Forensic Sci. Int. Digit. Investig.* **2022**, *42*, 301437. [CrossRef]
34. Sundresan, P.; Norita, N.; Valliappan, R. Internet of Things (IoT) digital forensic investigation model: Top-down forensic approach methodology. In Proceedings of the 2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC), Sierre, Switzerland, 7–9 October 2015; IEEE: New York, NY, USA, 2015; pp. 19–23.
35. Zia, T.; Liu, P.; Han, W. Application-specific digital forensics investigative model in internet of things (iot). In Proceedings of the 12th International Conference on Availability, Reliability and Security, Calabria, Italy, 29 August–1 September 2017; pp. 1–7.
36. Zulkipli, N.H.N.; Alenezi, A.; Wills, G.B. IoT forensic: Bridging the challenges in digital forensic and the internet of things. In Proceedings of the International Conference on Internet of Things, Big Data and Security, Porto, Portugal, 24–26 April 2017; SCITEPRESS: Setúbal, Portugal, 2017; pp. 315–324.
37. Engelhardt, F.; Güneş, M. A/sys Filesystem for the Internet of Things. In Proceedings of the NOMS 2022–2022 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 25–29 April 2022; IEEE: New York, NY, USA, 2022; pp. 1–6.
38. Matt, J.; Morris, S. Purple dawn: Dead disk forensics on Google's Fuchsia operating system. *Forensic Sci. Int. Digit. Investig.* **2021**, *39*, 301269.
39. Sandvik, J.P.; Franke, K.; Abie, H.; Arnes, A. Coffee forensics—Reconstructing data in IoT devices running Contiki OS. *Forensic Sci. Int. Digit. Investig.* **2021**, *37*, 301188. [CrossRef]