ISSN 2007-9737

PAREX: A Novel exFAT Parser for File System Forensics

Gaurav Gogia*, Parag Rughani

National Forensic Sciences University,
India

gaurav-gogia@outlook.com, parag.rughani@gmail.com

Abstract. File systems, being one of the core components of any computational device, contain the most important information which makes them pivotal to any digital forensics investigation. However, file system parsing is a complex process. Existing file system forensic software are capable of processing large datasets but often at the cost of either performance or resource utilisation. Slow evidence processing has a direct impact on investigation time, while higher resource requirements have a monetary impact. Digital Forensics labs are often on a constrained budget in terms of both time and money. So, they often need to define priorities on a case-by-case basis. Another major concern for forensic investigators is correctness. Tools that suffer from memory management issues may generate inconsistent reports or worse yet, increase overall attack surface for malware that may pollute investigator's workstation. This research proposes a novel open-source exFAT file system parsing library. It has been validated against the current open-source state-of-the-art: The Sleuth Kit (TSK), on a dataset of disk images ranging from 1MiB to 1TiB. Experimental results indicate that the proposed tool is 40 times faster and 17 times more memory efficient than TSK.

Keywords. ExFat, parsers, file system forensics, digital forensics.

1 Introduction

File systems are blueprints that provide an arrangement for operating systems to efficiently & reliably store all the files. Modern file systems can scale up to billions of files with each file reaching sizes over multiple terabytes. Such features have made file systems the de facto structure of storing data in secondary storage.

Some of the most common file systems include NTFS, exFAT, EXT4, APFS, and XFS. As the default storage method, most of the data is stored in secondary storage. This effectively turns file systems into a gold-mine of artefacts for digital forensics investigators [28].

To extract these artefacts, numerous digital forensic tools have been developed [36]. Almost all modern digital forensic tools have a file system parsing module in them. A file system parser is a tool that understands the structure of the file system it has to read. The parser must understand all the pre-defined fields in the file system and read them while understanding their specific meanings.

Some parts of the file system contain meta-data while others contain contents of the files stored in them. A file system is similar to a JavaScript Object Notation (JSON) object or an Extensible Markup Language (XML) object in the sense that all three of them present different ways to organize data.

Almost all of the modern digital forensic tools have file system parsing capabilities. Some of these modern tools include TSK [41] & EnCase [32]. These forensics tools parse the file system to extract the artefacts and run analytics on top of them. However, file system parsing can be a complex process.

While digital forensics tools are able to handle terabytes of data, they are often slow to process the large volume of data [29]. Processing performance and resource utilisation are inversly related, high performance processing requires higher memory in most cases. For instance, parsing a file system with a large number of files and directories can result in high memory usage, which can slow down the parsing process.

ISSN 2007-9737

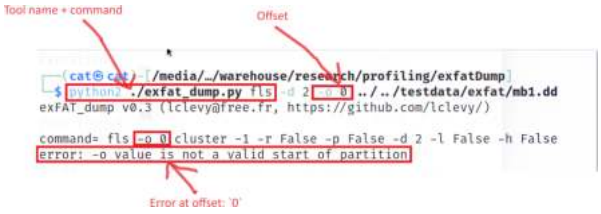

Fig. 1. exFAT file system not detected at offset '0'

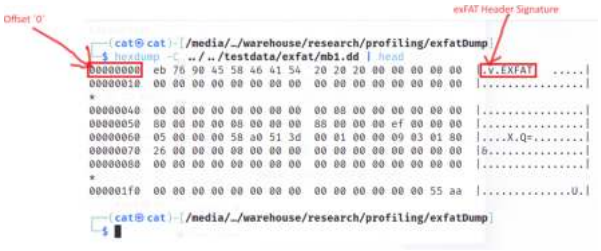

Fig. 2. exFAT file system header signature at offset '0'

When file system parsing is slow or inefficient, it slows down investigations. In certain instances, the indexing of an entire disk image with millions of files can be a time-consuming process, particularly when dealing with legacy equipment. The "Experiment" section of this research offers empirical evidence derived from testing various scenarios on five different platforms.

This research focuses on open-source file system parsers. Specifically designed for the exFAT file system [2], aiming to contribute an efficient and fast exFAT parser. The exFAT file system was selected for its compatibility with most of the major electronic devices [20, 23, 43, 39, 40, 1]. The "Related Work" section provides a more comprehensive discussion of this aspect.

1.1 Problem Statement

Forensic investigations often suffer from large backlog of cases due to slow, bulky, and inefficient digital forensic tools [29]. Digital evidence is relevant in about 90% of the cases [26]. The amount of data that must be acquired, analyzed, indexed, processed, triaged, and reported, keeps increasing every passing day. Storage devices have evolved to preserve multiple terabytes of data.

While advanced forensic tools and high-powered hardware tools are available, a lot of digital forensics labs don't have the budget for these advanced tools [17]. Building a digital forensics lab can easily cost over $100,000 [33], this cost is excluding yearly maintenance cost of the lab. Labs that rely on open-source tools find themselves lagging behind because existing tools are slow and memory in-efficient.

Next sub-section will explain the current state of open source exFAT file system parsers for digital forensic tasks. According to a joint report by Google and Microsoft, a staggering 70% of all security bugs can be traced back to memory safety issues [47, 48]. Although there are other open-source tools available for file system parsing, such as dfir_ntfs [30] & exfatDump [18] written in Python [37], they appear to have limited functionality. For instance, dfir_ntfs, is designed for parsing the Volume Boot Record (VBR), Master Boot Record (MBR) structures, and file metadata without the ability to read or extract any entries.

While, exfatDump, published in October 2015, unfortunately, seems to be non-operational. As shown in Figure 1, exfatDump fails to identify the start of the exFAT file system at the specified offset 0, which is indicated using hexdump [16] tool in figure 2. In contrast, the proposed solution of this research offers a robust and comprehensive approach to parsing file system structures.

1.2 Contribution

The primary contribution of this research is LIBXFAT, an exFAT file system parsing library [3]. Additionally, a secondary add-on to the contribution is PAREX, a Command Line Interface (CLI) tool built on top of LIBXFAT [4]. PAREX offers four options for parsing disk images, which are further explained in Table 1.

The main features of these software tools include listing all entries in the file system, including deleted and deleted entries, as well as extracting file contents in a forensically sound manner. Both LIBXFAT and PAREX were developed using the Go programming language [14], which was selected for its simplicity, easy concurrency, and built-in memory safety features.

Table 1. PAREX options

Parsing Option	Operation
0	List Root Entries
1	List All Entries - With Metadata
2	List All Entries - With Count
3	Extract All Entries - Collected
4	Extract All Entries - Recursively

Table 2. Indexed files per image

Image Size	Indexed Files
1 MiB	7
512 MiB	4540
1 GiB	11336
10 GiB	240295
32 GiB	442
64 GiB	313030

The memory safety aspect of the Go has been validated by Felix A. Wolf et al., as cited in reference [45]. To ensure the correctness of LIBXFAT, its results were compared with those obtained from TSK and Autopsy [5]. Furthermore, for benchmarking purposes, LIBXFAT was profiled against TSK using various parameters. Detailed explanations of these experiments can be found in the experiments section.

1.3 Outline

The rest of the paper is organized in 5 major sections: Section 2 ”Related Work” discusses some related research work with the current state of file system parsing tools. Section 3 ”Background” explains the structure of exFAT file system. This section can be skipped by those readers who understand this file system well.

Section 4 “Experiment” presents experiments, experimental methodology, and the data generated during those experiments. In Section 5 “Discussion”, experimental results are interpreted; Section 6 “Conclusion” concludes and discusses future research work and potential applications of this work.

2 Related Work

This section reviews studies on file system forensics, highlights the importance of exFAT from a forensic perspective, and explores existing software solutions. It focuses on solutions that offer API/Library integration for developers to promote inter-operability in the field.

File System Forensics: The significance of file system forensics is paramount; virtually all digital artefacts can be traced back to the file system. A recent paper explored the application of machine learning algorithms for identifying contraband within file systems [28]. Extensive research has been conducted regarding forensic and anti-forensic techniques for various file systems.

One such study proposed a novel file recovery algorithm designed to recover deleted files from the FAT32 file system [8]. Another research introduced a scheme aimed at detecting data in FAT32 file systems that leaves no traces [46]. However, investigating file systems isn't solely limited to recovery files or identifying data streams. For instance, ExtSFR is a scalable file recovery framework that is compatible with EXT file systems [19].

APFS, another crucial file system, has been sparsely studied due to its proprietary, closed-source nature. Researchers interested in developing file recovery algorithms for this file system had to resort to reverse engineering [36]. Similarly, the proprietary Resilient File System (ReFS) also necessitates reverse engineering to extract valuable information [32].

These efforts pose various legal and technical questions around the process of reverse engineering a file system [42]. However, reverse engineering and file recovery are not the sole use cases in file system forensics. Another scenario involves the simple reading and classification of files.

This has spurred research into classification algorithms, with some based on neural networks that traverse the file system to identify contraband [27]. File system forensics has indeed spurred a numerous research initiatives. To that end, this paper will primarily focus on the exFAT file system.

Table 3. exFAT region layout with offset & size in sectors

Sub-Region Name	Offset (Hex)	Size (Decimal)
Main Boot Region		
Main Boot Sector	0x0	1
Main Extended Boot Sectors	0x1	8
Main OEM Parameters	0x9	1
Main Reserved	0xA	1
Main Boot Checksum	0xB	1
Backup Boot Region		
Backup Boot Sector	0xC	1
Backup Extended Boot Sectors	0xD	8
Backup OEM Parameters	0x15	1
Backup Reserved	0x16	1
Backup Boot Checksum	0x17	1
FAT Region		
FAT Alignment	0x18	FatOffset – 24
First Fat	FatOffset	FatLen
Second Fat	FatOffset + FatLen	FatLen * (FatCount – 1)
Data Region		
Cluster Heap Alignment	FatOffset + FatLen * FatCount	ClusterHeapOffset – (FatOffset + FatLen * FatCount)
Cluster Heap	ClusterHeapOffset	ClusterCount * 2
Excess Space	ClusterHeapOffset + ClusterCount*2	VolumeLen – (ClusterHeapOffset + ClusterCount * 2)

exFAT Forensics: The exFAT file system has emerged as the de facto standard for removable storage devices and those utilizing NAND flash storage technology, including thumb drives, SDXC cards, eMMC storage in laptops, and more.

A key factor driving its widespread adoption is the deliberate design choice made by its creators—minimising write operations to promote the longevity of storage devices. Another pivotal aspect is its compatibility with major operating systems such as Windows, MacOS, Ubuntu, Android, and other Unix-based systems [15]. Forensic workstations leverage exFAT file systems to ensure seamless interoperability across different operating systems [20].

Despite its popularity, the algorithms used to parse and analyse devices employing the exFAT system are not publicly available, posing a challenge to comprehensive forensic analysis [23]. The significance of the exFAT file system is underscored by Yves Vandermeer et al., who conducted an in-depth study on its data structure [43]. Furthermore, various studies highlight the use of exFAT file systems in a diverse array of devices, from medical equipment to drones [39, 40, 1]. Consequently, advancing our understanding and capabilities in exFAT forensics holds paramount importance.

Open Source Digital Forensics: Several digital forensics software programs capable of parsing the exFAT file system currently exist.

Table 4. Number of entires per disk image

Size	All	Root	Indexed	Deleted
1 MiB	10	6	10	0
512 MiB	4734	19	4733	1
1 GiB	11580	20	11561	19
5 GiB	1	9	9	0
10 GiB	246910	13	246845	65
25 GiB	11	11	11	0
32 GiB	556	13	556	0
40 GiB	20	20	20	0
64 GiB	325517	112	324865	652
128 GiB	533292	150	532444	848
256 GiB	1375122	152	1374820	302
500 GiB	45	45	45	0
512 GiB	2550344	152	2550212	132
1 TiB	5057669	1546	5053570	4099

However, the process of validating, benchmarking, and developing trust in these tools presents a considerable challenge [7]. Although open-source file system forensics software are available [41, 40], their over-reliance on memory-unsafe programming languages such as C & C++ can cause supply chain security challenges.

Multiple entities, including Google and Microsoft, along with independent researchers, have reported that over 70% of memory-related flaws in operating systems and browsers can be directly attributed to code written in C or C++ [47, 48, 49, 12].

These memory-related bugs, if exploited, can compromise a digital forensics workstation causing damage to all the cases being investigated on the same machine or on the same local network, depending on the extent of the exploit.

The Go programming language, developed by Google, offers a safer alternative [45]. Its design goals are simplicity, ease of development, and high performance. When compared to other memory-safe programming languages, the performance of Go is comparable to C++ [35, 11, 22, 34].

However, only one open-source library has been identified to date [10] that is written in Go programming language, but it lacks features such as recursive file system traversal and access to metadata information like the number of clusters a file contains.

This paper introduces a new open-source library and a CLI tool for parsing the exFAT file system. This library has been validated with and benchmarked against industry standard tools. Validation and benchmark tests are explained in more detail in Section 4 ”Experiment” section.

3 Background

The exFAT file system was developed by Microsoft to address the limitations of the FAT32 file system, particularly in relation to flash storage devices like SD cards. The exFAT file system comprises three primary regions: the Boot Region (also known as the superblock), the FAT Region, and the Data Region. The Boot Region, occupying the first 512 bytes, contains crucial metadata and initialization data, including the file system signature and parameters.

The FAT Region, as the name suggests, stores the file allocation table, which manages file and directory locations. The Data Region stores the actual contents and metadata of all the files and folders. A comprehensive study by Julian Heeger et al. provides detailed insights into the architecture and functioning of the exFAT file system [15]. Table 3 explains subsections of these regions with their offsets [2].

4 Experiment

To evaluate the correctness and performance for PAREX software (powered by LIBXFAT), functional and benchmarking experiments were carried out. For benchmarking experiments FLS software (powered by TSK) was used as the control/standard software against all the comparison was made. The upcoming subsections will provide detailed explanations of the experiment setup, the experiment itself, and the corresponding results.

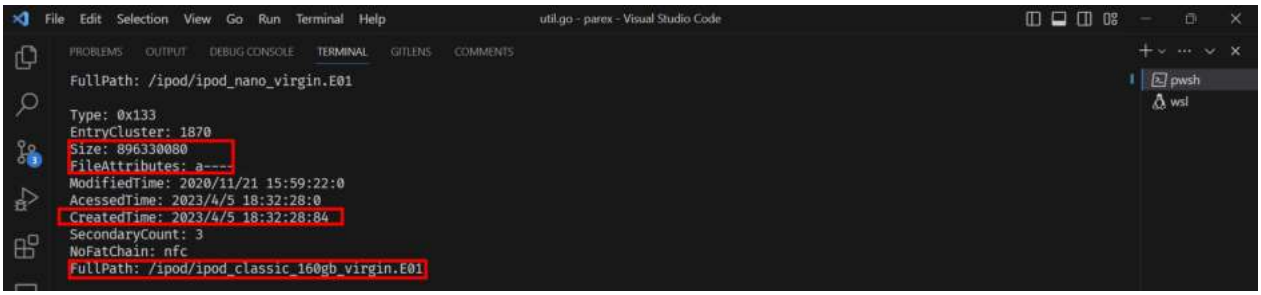

Fig. 3. Verify metadata - PAREX

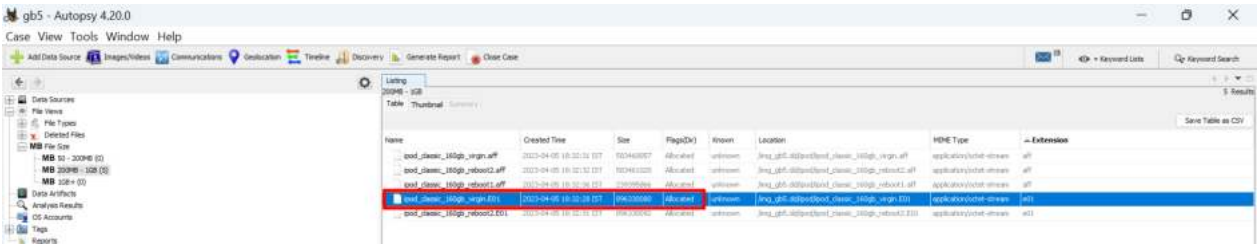

Fig. 4. Verify metadata - autopsy

4.1 Experiment Setup

To validate both PAREX and FLS, the first step involved generating a dataset of raw disk images. Most of the disk images were generated synthetically by creating several files and folders in an external storage device. Additionally, a portion of the dataset was downloaded from NIST CFReDS repositories [31].

The downloaded files were saved into a separate partition in a Virtual Machine. Raw disk images were created out of the partitions where the files were downloaded. Disk images ranging from 1MiB to 1TiB were created to cover a range of scenarios. Experiment setup is divided into two parts: Functional Tests Setup & Benchmark Tests Setup.

They explain the experiment environment and profiling methods used to conduct the experiment. Within the context of this study, the terms 'entry' & 'entries' are employed to denote an 'entry' & a number of 'entries' within the file system. This term is used universally to represent both files and folders. Table 4 exhibits the total number of entries, root entries, indexed entries, and deleted entries for a disk image, along with their respective sizes.

4.1.1 Functional Test Setup

The experimental environment for the functional tests was intentionally simple. A single platform was chosen to evaluate the basic functionality of PAREX software developed within this research. Performance assessments of all the tools were executed using GNU/Time software [13] within a Windows Subsystem for Linux 2 (WSL-2) environment [6].

4.1.2 Benchmark Test Setup

The benchmark tests were performed in five different environments: WSL-2 [6], Anarchy Linux [9], Windows 10 Professional [24], Windows 11 Professional [25], and Kali Linux [21]. These environments were created using VMWare Workstation Pro 17 [44].

Windows 10, Windows 11, Anarchy Linux, & Kali Linux environments were created with 4 vCPUs and 6 GiB RAM. A 12 TiB 7200RPM external HDD was used for storage. While on the other hand, WSL-2 had 16 vCPUs, 32 GiB RAM, and 1 TiB internal M2 SSD storage drive.

Algorithm 1 Process Profiling Algorithm

start_time ← current time
p ← Process object for proc.pid
Initialise empty lists: cpu_use_list, mem_use_list, thread_count_list, read_bytes_list, write_bytes_list
while proc is running **do**
    cpu ← CPU percent of p divided by total number of CPUs
    Append cpu to cpu_use_list
    mem ← memory used by p
    Append mem to mem_use_list
    Append number of threads used by p to thread_count_list
    Get I/O counters for p
    Append read bytes and written bytes to read_bytes_list and write_bytes_list respectively
    Sleep for sampling_rate seconds
end while
end_time ← current time
avg_cpu_use ← Average of cpu_use_list
avg_ram_use ← Average of mem_use_list
avg_thread_count ← Average of thread_count_list
avg_io_read ← Average of read_bytes_list
avg_io_write ← Average of write_bytes_list
return avg_cpu_use, avg_ram_use, avg_thread_count, avg_io_read, avg_io_write

To profile all these experiments, a python [37] script was written using psutil [38] library. This library helps in profiling system events like memory use, execution time, thread count, processor use, disk reads/writes etc.

The version of the library at the time of writing this paper is v5.9.5. This library has a proven record and is being actively maintained on GitHub by many contributors. Algorithm 1 explains the profiling script.

4.2 Functional Tests

For the functional tests, all the files were extracted out of the disk images using PAREX to validate the correctness of the CLI tool. Table 2 shows image size and the number of indexed files per disk image.

To verify the correctness of PAREX, results of data parsing and extraction were compared with the results of TSK & Autopsy. Please note that an additional command in PAREX was executed for this experiment to list out all the files and their metadata out of the disk image.

Figures 3 & 4 represent matching metadata between PAREX & Autopsy to solidify the correctness of exFAT file system parsing in PAREX. On the other hand, figure 7 represents matching SHA-256 hash of the extracted files by PAREX & TSK illustrating correctness of file extracted by PAREX.

Mean and Standard Deviation out of experiment data was calculated to compare performance, efficiency, and consistency between PAREX and FLS. Upcoming sub-sections will delve into the details of the experiemnts and present results.

4.2.1 List Root Entries

In the first benchmark test, FLS was executed in its default state without any flags, while PAREX(powered by LIBXFAT) was run with the 0 option. This option parses root dir entry and returns them for the user, please find table 1 for more details. This approach ensured that both tools used a 0 offset of the disk image as the starting point and exclusively returned root entries from the disk image.

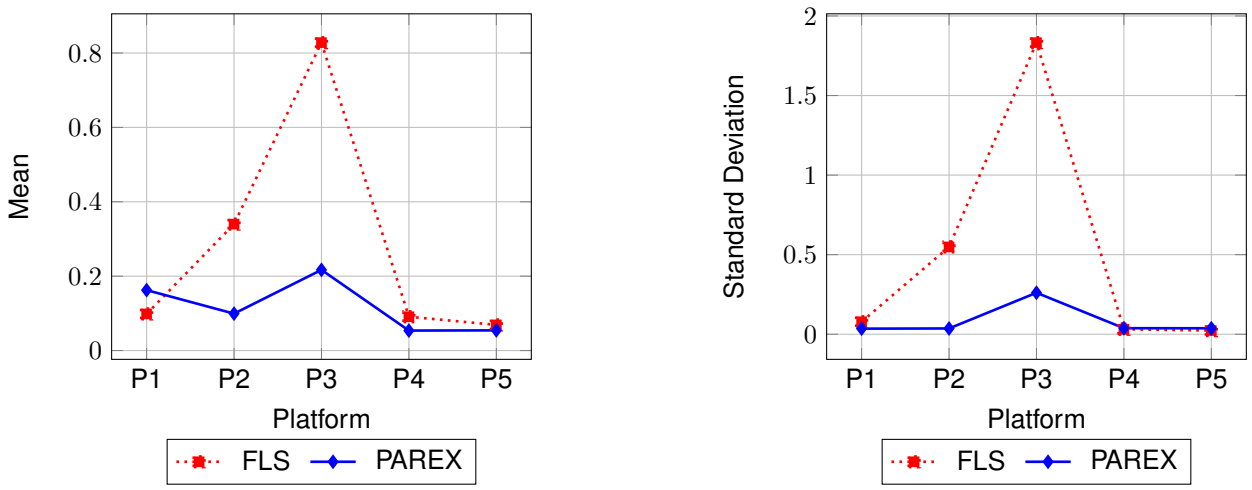

Fig. 5. List root entries — execution time (seconds) — lower is better

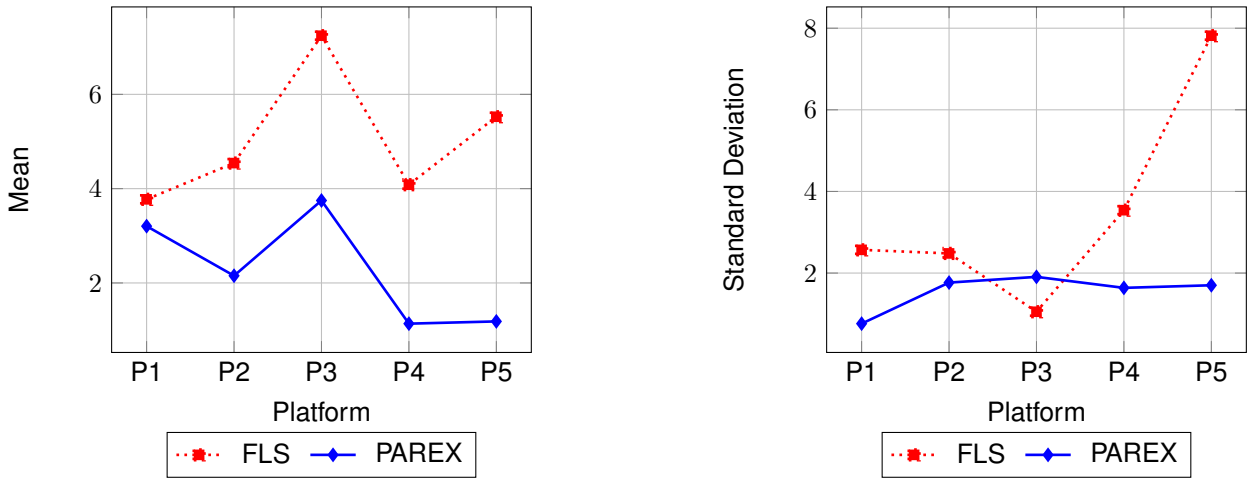

Fig. 6. List root entries — RAM use (MB) — lower is better

4.2.2 List All Entries

In the second benchmark test, FLS was run with options '-u -r' while PAREX was run with option '2'. This option lists out all the indexed entries in the file system while keeping track of count of number of entries, please find table 1 for more details. This approach ensured that both tools used a 0 offset of the disk image as the starting point and exclusively returned all the indexed endtries from the disk image.

4.3 Experiment Results

Mean & Standard Deviation was calculated for all the statistical data that was acquired by profiling the experiments run to benchmark both PAREX and FLS software tools. Results of these experiments have been visualized to clearly state the difference in performance, efficiency, and consistency between the two tools. Figures 5, 6, 8, and 9 represent comparative analysis between PAREX and FLS on various platforms through mean execution time, standard

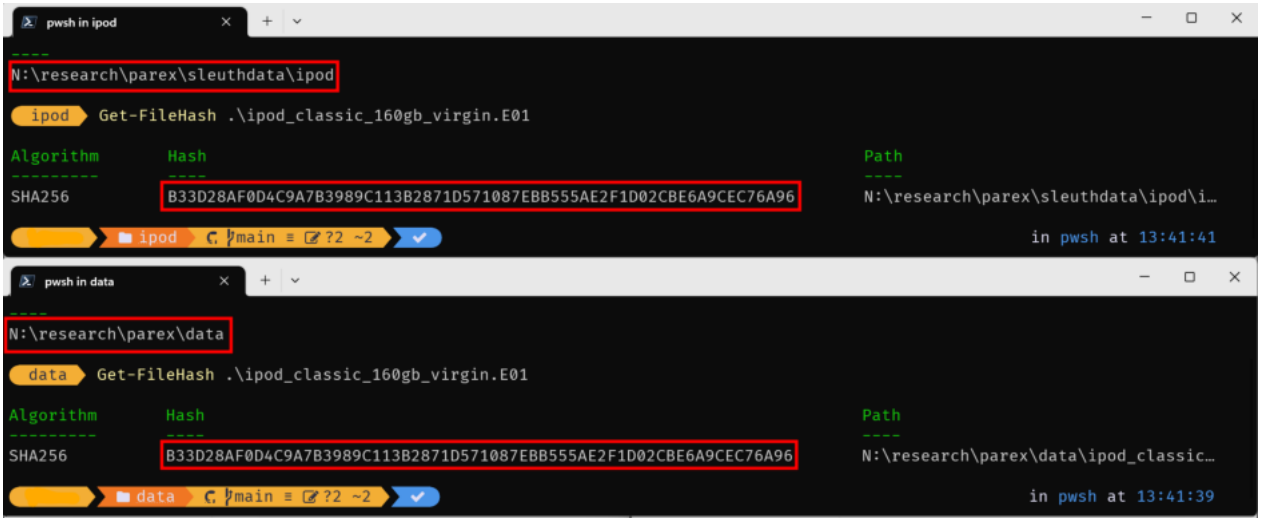

Fig. 7. Verify file extraction

deviation in execution time, mean RAM use, and standard deviation in RAM use. The following list elucidates the platforms on which these experiments were conducted:

– P1 - WSL 2,
– P2 - Windows 10,
– P3 - Windows 11,
– P3 - Arch Linux,
– P4 - Kali Linux.

4.4 Caveats

This study has several important caveats that must be considered when interpreting its findings. Prrofiling in a Windows Subsystem for Linux 2 (WSL-2) environment is currently not fully reliable. The high level of abstraction that WSL-2 introduces an inherent challenge in accurately profiling all parameters.

This limitation potentially impacts the precision and consistency of our results obtained from this environment. To alleviate this concern, identical experiments were conducted on Anarchy Linux, Kali Linux, Windows 10, and Windows 11 virtual machines were conducted.

The diverse range of environments helps in establishing a comprehensive and more reliable picture of the software's execution time. Secondly, the accuracy of profiled data is inversely proportional to the sampling rate.

As the sampling rate decreases, the chances of obtaining accurate profiling data diminishes. This phenomenon occurs because lower sampling rates have a reduced ability to capture all system state changes accurately. To examine this effect and capture data at different levels of granularity, we performed profiling at sampling rates of 0ms, 50ms, and 100ms.

Lastly, the accuracy of profiled data can also be affected if an operation runs faster than the sampling rate. This discrepancy can lead to the operation being entirely missed by the profiler, especially for those operations that complete within a time frame shorter than the sampling rate. This occurrence introduces another potential source of error in the profiling data.

Therefore, it's critical to understand that the results of this study are subjected to these inherent limitations of the profiling process. Future work could focus on developing methods to mitigate these issues and enhance the accuracy of profiling data.

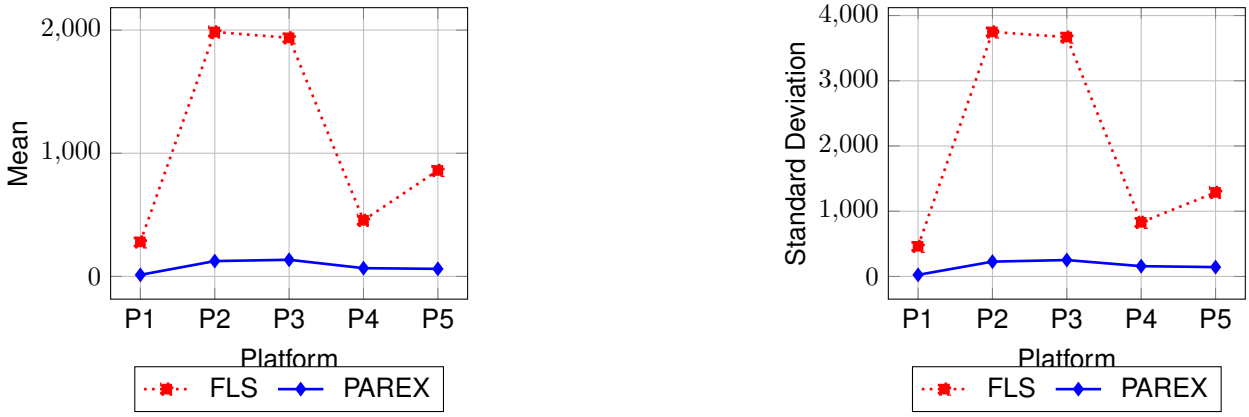

Fig. 8. List all entries — execution time (seconds) — lower is better

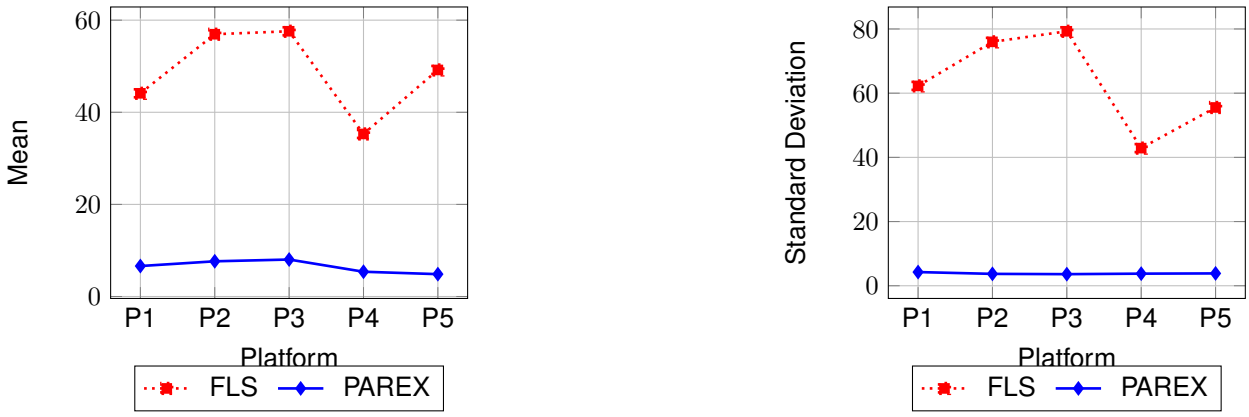

Fig. 9. List all entries — RAM use (MB) — lower is better

5 Discussion

This section presents the outcomes of functional and benchmark tests.

The experimental results indicate that the PAREX and LIBXFAT library accurately parses the exFAT file system, effectively identifying root entries, traversing all directories and sub-directories to locate the remaining entries, and extracting file content in a forensically sound manner.

The benchmark tests reveal that the PAREX, is significantly faster and more memory efficient than FLS. Experiments also reveal that PAREX software performs much more consistently across different platforms.

This is another important finding, as it means that PAREX can be used to process large exFAT formatted devices reliably with high speed while keeping minimum memory footprint on the investigator's workstation. However, this research does have a limitation: the absence of active detection for deleted entries.

While the PAREX can identify obviously deleted or deleted entries evidenced by 0x0 listed as the entry cluster offset, it does not carry out any advanced operations to detect less obvious deleted entries.

Furthermore, PAREX does not employ any statistical or pattern matching techniques to identify deleted directory entries, it can only detect deleted file entries.

These features can be added in future to arm PAREX with more features for forensic artefact analyses. Overall, the findings of the research are positive. PAREX is a promising tool for recovering data from exFAT formatted devices. However, future work should focus on addressing the limitation of the research by developing methods to actively detect deleted entries.

6 Conclusion

In this study, an open-source library and a CLI tool were developed for parsing the exFAT file system. To validate correctness and performance, deep profiling and benchmarking tests were conducted using the psutil library on five different platforms: WSL-2, Anarchy Linux, Kali Linux, Windows 10, & Windows 11. The developed tools were benchmarked against industry standard open-source tools: The Sleuth Kit (TSK) & Autopsy. The results demonstrate that the developed tools are over 40 times faster than the control set while also being 17 times more memory efficient.

The developed software consistently present effective and efficient results over multiple platforms. These results directly impact the cost of acquisition and maintenance of workstations and other associated computer hardware, be it on-premises or on cloud. However, to further enhance the software, several optimization strategies can be implemented.

These include improving the handling of multiple goroutines, implementing thread-pooling for larger objects, and conducting deeper profiling tests to identify and eliminate unnecessary object allocations and deallocations. Moreover, future research prospects involve addressing the limitation of active deleted file detection and deleted file recovery by developing additional features.

References

1. **Allen-Barton, T. E., Bin-Azhar, M. A. H. (2018).** Open source forensics for a multi-platform drone system. Computing, Digital Forensics and Cybersecurity, pp. 83–96. DOI: 10.1007/978-3-319-73697-6_6.
2. **alvinashcraft (2022).** Exfat file system specification - win32 apps. https://learn.microsoft.com/en-us/windows/win32/fileio/exfat-specification.
3. **aoiflux (2023).** Libxfat. https://github.com/aoiflux/libxfat.
4. **aoiflux (2023).** Parex. https://github.com/aoiflux/parex.
5. **Autopsy (2023).** Digital forensics. https://www.autopsy.com/.
6. **Barnes, H. (2021).** Pro Windows subsystem for Linux (WSL). Apress. DOI: 10.1007/978-1-4842-6873-5.
7. **Bhat, W. A., AlZahrani, A., Wani, M. A. (2021).** Can computer forensic tools be trusted in digital investigations?. Science & Justice, Vol. 61, No. 2, pp. 198–203. DOI: 10.1016/j.scijus.2020.10.002.
8. **Chen, B., Guan, J., Wang, H., Yao, G. (2021).** A novel data recovery algorithm for FAT32 file system. 2021 2nd International Conference on Information Science and Education (ICISE-IE), pp. 605–608. DOI: 10.1109/ICISE-IE53922.2021.00143.
9. **Community, A. L. (2023).** Anarchy Installer. https://anarchyinstaller.gitlab.io/.
10. **dsoprea (2022).** go-exfat. https://github.com/dsoprea/go-exfat.
11. **Fua, P., Lis, K. (2020).** Comparing python, go, and C++ on the n-queens problem. http://arxiv.org/abs/2001.02491. DOI: 10.48550/arXiv.2001.02491.
12. **Gao, Y., Chen, L., Shi, G., Zhang, F. (2018).** A comprehensive detection of memory corruption vulnerabilities for C/C++ programs. 2018 IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications

(ISPA/IUCC/BDCloud/SocialCom/SustainCom), pp. 354–360. DOI: 10.1109/BDCloud.2018.00062.

13. **GNU (2023).** GNU time - GNU project - free software foundation. https://www.gnu.org/software/time/.
14. **Google (2023).** The go programming language. https://go.dev/.
15. **Heeger, J., Yannikos, Y., Steinebach, M. (2022).** An introduction to the exFAT file system and how to hide data within. Journal of Cyber Security and Mobility, Vol. 11, No. 2, pp. 239–264. DOI: 10.13052/jcsm2245-1439.1125.
16. **Kerrisk, M. (2023).** hexdump(1) - Linux manual page. https://www.man7.org/linux/man-pages/man1/hexdump.1.html.
17. **Lawrence, T., Karabiyik, U., Shashidhar, N. (2018).** Equipping a digital forensic lab on a budget. 2018 6th International Symposium on Digital Forensic and Security (ISDFS), pp. 1–7. DOI: 10.1109/ISDFS.2018.8355345.
18. **Iclevy (2023).** An experimental tool for forensic analysis of ExFAT filesystem. https://github.com/lclevy/exfatDump.
19. **Lee, S., Jo, W., Eo, S., Shon, T. (2020).** ExtSFR: Scalable file recovery framework based on an Ext file system. Multimedia Tools and Applications, Vol. 79, No. 23, pp. 16093–16111. DOI: 10.1007/s11042-019-7199-y.
20. **Lin, X. (2018).** Building a forensics workstation. Introductory Computer Forensics: A Hands-on Practical Approach, pp. 53–89.
21. **Linux, K. (2024).** Penetration testing and ethical hacking Linux distribution. https://www.kali.org/.
22. **Lion, D., Chiu, A., Stumm, M., Yuan, D. (2022).** Investigating managed language runtime performance: why javascript and python are 8x and 29x slower than c++, yet java and go can be faster? Usenix ATC'22, pp. 835–852.
23. **Mason, S., Seng, D. (2017).** Electronic evidence. University of London Press. DOI: 10.14296/517.9781911507079.
24. **Microsoft (2023).** Download Windows 10. https://www.microsoft.com/en-in/software-download/windows10.
25. **Microsoft (2023).** Download Windows 11. https://www.microsoft.com/en-in/software-download/windows11.
26. **Miller, C. M. (2022).** A survey of prosecutors and investigators using digital evidence: A starting point. Forensic Science International: Synergy, Vol. 6, pp. 100296. DOI: 10.1016/j.fsisyn.2022.100296.
27. **Mohammad, R. M. (2018).** A neural network based digital forensics classification. 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA), pp. 1–7. DOI: 10.1109/AICCSA.2018.8612868.
28. **Mohammad, R. M. A., Alqahtani, M. (2019).** A comparison of machine learning techniques for file system forensics analysis. Journal of Information Security and Applications, Vol. 46, pp. 53–61. DOI: 10.1016/j.jisa.2019.02.009.
29. **Montasari, R., Hill, R. (2019).** Next-generation digital forensics: Challenges and future paradigms. 2019 IEEE 12th International Conference on Global Security, Safety and Sustainability (ICGS3), pp. 205–212. DOI: 10.1109/ICGS3.2019.8688020.
30. **msuhanov (2023).** An NTFS/FAT parser for digital forensics & incident response. https://github.com/msuhanov/dfir_ntfs.
31. **NIST (2023).** CFReDS Portal. https://cfreds.nist.gov/.
32. **Nordvik, R., Georges, H., Toolan, F., Axelsson, S. (2019).** Reverse engineering of ReFS. Digital Investigation, Vol. 30, pp. 127–147. DOI: 10.1016/j.diin.2019.07.004.
33. **Opolskii, V., Stupina, M. (2021).** Consumer-grade storage comparative

analysis in the context of digitalization of the agro-industrial complex. IOP Conference Series: Earth and Environmental Science, Vol. 937, No. 3, pp. 032079. DOI: 10.1088/1755-1315/937/3/032079.

34. **Pascal, C., Hurt, I., Mattson, T. G. (2022).** Towards a graphblas implementation for go. 2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 01–04. DOI: 10.1109/IPDPSW55747.2022.00052.
35. **Pascal, C., Herzeel, C., Verachtert, W. (2019).** Comparing ease of programming in C++, go, and java for implementing a next-generation sequencing tool. Evolutionary Bioinformatics, Vol. 15. DOI: 10.1177/1176934319869015.
36. **Plum, J., Dewald, A. (2018).** Forensic APFS file recovery. Proceedings of the 13th International Conference on Availability, Reliability and Security, pp. 1–10. DOI: 10.1145/3230833.3232808.
37. **Python (2023).** Welcome to Python.org. https://www.python.org/.
38. **Rodola, G. (2023).** Psutil: Cross-platform lib for process and system monitoring in Python. https://github.com/giampaolo/psutil.
39. **Schmitt, V. (2022).** Medical device forensics. IEEE Security & Privacy, Vol. 20, No. 1, pp. 96–100. DOI: 10.1109/MSEC.2021.3127490.
40. **Senturk, S., Apaydın, T., Yasar, H. (2020).** Image and file system support framework for a digital mobile forensics software. 2020 Turkish National Software Engineering Symposium (UYMS), pp. 1–3. DOI: 10.1109/UYMS50627.2020.9247055.
41. **sleuth kit, T. (2022).** The sleuth kit (Tsk) & autopsy: Open source digital forensics tools. https://sleuthkit.org/.
42. **Stoykova, R., Nordvik, R., Ahmed, M., Franke, K., Axelsson, S., Toolan, F. (2022).** Legal and technical questions of file system reverse engineering. Computer Law & Security Review, Vol. 46, pp. 105725. DOI: 10.1016/j.clsr.2022.105725.
43. **Vandermeer, Y., Le-Khac, N. A., Carthy, J., Kechadi, T. (2018).** Forensic analysis of the exFAT artefacts. DOI: 10.48550/arXiv.1804.08653.
44. **VMWare (2023).** Download VMware Workstation Pro. https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html.
45. **Wolf, F. A., Arquint, L., Clochard, M., Oortwijn, W., Pereira, J. C., Müller, P. (2021).** Gobra: Modular specification and verification of go programs. Computer Aided Verification, pp. 367–379. DOI: 10.1007/978-3-030-81685-8_17.
46. **Xu, S., Liu, F., Meng, L., Wang, L., Chang, X., Yang, W. (2022).** A scheme of traceless file deletion for windows FAT32 file system. Proceedings of the 2021 ACM International Conference on Intelligent Computing and its Emerging Applications, pp. 89–93. DOI: 10.1145/3491396.3506515.
47. **ZDNET (2022).** Chrome: 70% of all security bugs are memory safety issues. www.zdnet.com/article/chrome-70-of-all-security-bugs-are-memory-safety-issues/.
48. **ZDNET (2022).** Microsoft: 70 percent of all security bugs are memory safety issues. www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/.
49. **Zhang, H., Wang, S., Li, H., Chen, T. H., Hassan, A. E. (2022).** A study of C/C++ code weaknesses on stack overflow. IEEE Transactions on Software Engineering, Vol. 48, No. 7, pp. 2359–2375. DOI: 10.1109/TSE.2021.3058985.

Article received on 12/01/2024; accepted on 11/03/2024.
**Corresponding author is Gaurav Gogia.*