**Developing File Extraction Methods for Open-Source Forensics Software in C#**

Mikolaj M. Mroz

School of Design and Informatics

Abertay University

DUNDEE, DD1 1HG, UK

# ABSTRACT

**Context**

OpenForensics is an open-source file carving tool for Windows created to showcase the advantages of implementing advanced GPU-based pattern-matching techniques in a digital forensics (DF) file-carving scenario. The tool demonstrates significant improvements over currently available software while searching through data but is limited by its ability to detect and reproduce the scanned files.

**Aim**

The aim of this project is to refactor OpenForensics with improved file-detection and validation algorithms to improve the program’s reliability and performance to reproduce files with accuracy in mind and as a result, make it more fit for purpose in a professional forensic analysis setting.

**Method**

Development will take place in a local Visual Studio (VS) 2022 environment on the latest available OpenForensics source code release (v1.85b) such that the files can be submitted to GitHub only once everything has been approved from communicating with the author. Research will be conducted into current implementations of file-detection algorithms across code sharing sites like GitHub, also analyzing the source code of the file-carving tool Foremost - which the current OpenForensics file-carving implementation is built on, as well as the original paper detailing the creation of the tool (Bayne, et al., 2018)

A standalone prototype file-detection algorithm will be created in C# to scan a single file in and then be gradually developed into a full implementation, being able to comprehend multiple files of different types in a secure and robust manner.

**Results**

The result of this project will be a refactored OpenForensics VS solution containing an improved file-detection algorithm capable of detecting a wide range of filetypes typically found in a DF investigation. Information regarding the development and testing of this software will be detailed in a dissertation format for understanding and reconstruction purposes, with the final revision of the software also being submitted to the original OpenForensics repository in the form of a pull request.

**Conclusion**

This project will demonstrate the process of developing file-detection techniques for Open-Source forensic software in C#,

aiming to integrate an improved file-detection algorithm into OpenForensics to provide a faster, more effective, and reliable tool for use in DF investigations.

**Keywords**

Digital Forensics, File-carving, Filetype, Refactor, Pattern-matching, File-detection, Header, Footer, Algorithm, Implementation, OpenForensics, Open Source

1. **INTRODUCTION**

The advent of digital storage and its subsequent surge in users around the world subjectively benefits many aspects of everyday living. As with most technology however, an increase in popularity also correlates with a rise in malicious activity. An estimated 97% of households in the UK own at least one personal computer (Alsop, 2019), therefore it should come as no surprise that computer memory and storage are now the most convenient way for attackers to facilitate illegal digital material, with an estimated 90% of crimes committed now featuring some sort of digital element (Davidson, et al. 2019) in DF investigations.

This radical shift in the types of evidence forensic agents must deal with in the modern days of technology has spurred a desperate need for dependable and efficient tools. The reliability of these tools and their performance is imperative, as they are used by forensics agents to produce results for court proceedings in criminal cases, in which any delay in analysis results could interfere with the judicial process. A 2015 report by the His Majesty's Inspectorate of Constabulary Services (HMIC) revealed that delays in the forensics process were often delayed by upwards of 12 months (HMIC, 2015), posing a huge risk for the integrity of digital evidence in the court of law. Hence, OpenForensics was created.

What sets this software apart from what is currently available on the market is its ground-breaking implementation of file carving technology utilizing the computer’s graphics processing unit (GPU) to search through metadata as opposed to using slower, CPU-based algorithms.

The issue lies with its file detection and reconstruction algorithm, which is unreliable when dealing with complex file types, impacting the credibility and reliability of its results. This forms the aim of the project: to refactor OpenForensics and introduce improved file detection techniques to work with a wider range of file types and to do so in a quick, memory-efficient manner with the goal of making this tool more viable for use in a professional context.

1. **BACKGROUND**

File carving works by importing a specified amount of data -typically the contents of a storage device - and scanning through it for headers, footers, and file contents, all of which uniquely identify the file and its data.

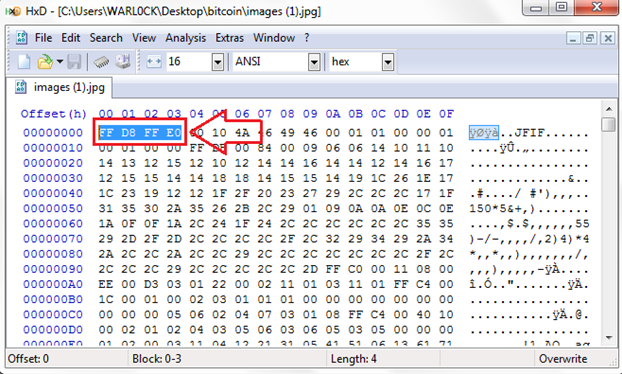


Figure 1 - The header of a JPG, commonly known to be FFD8FFE0 when examined in a HEX editor. (Warlock, 2018)

There are, however, several commonly utilised algorithms that would improve the file-validation capabilities of the program. These differ greatly in the sort of information they read in and process and allow for a more robust mechanism for determining the necessary details associated with different types of files.

**2.1 Header-Footer**

This is the type currently in use in OpenForensics. Header-Footer algorithms seem robust in principle, only reading in the data stored between a header (where a file’s type is determined) and its end, not reading in any excess data.

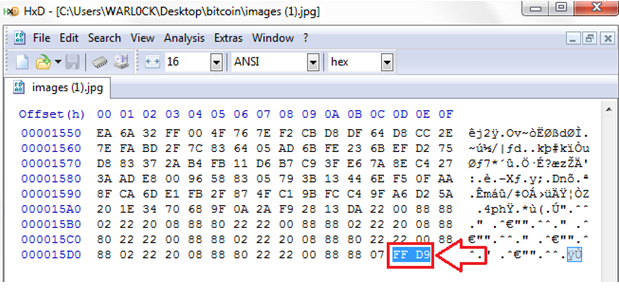


Figure 2 - The footer of a JPG, commonly known to be FFD9 as seen in a HEX editor. Header-size algorithms would not consider this during processing. (Warlock, 2018)

Unfortunately, this algorithm struggles greatly with nestled filetypes, which may hide additional information. This is most commonly an issue with .ZIP files as documented by Dr. Ethan Bayne but can also impact image, music, and document files which can sometimes nest hidden thumbnail images and other files that can interfere with identification.

* 1. **File Structure-based**

This method takes the internal structure of the file into consideration during processing. This allows it to reassemble fragmented files like HTML files where the header-footer algorithm fails, analysing the contents of the read files in a process called “Semantic Carving”, named so because of its knowledge to reconstruct a file based on its format.

**2.3 SmartCarving**

SmartCarving is another file carving technology that aims to reconstruct fragmented files. Developed for the Digital Forensic Research Workshop (DFRWS) forensic challenge by the founders of Digital Assembly, SmartCarving looks at the header block of a file to determine its format, and then analyses any following blocks of data to see if they relate to the starting block. This then results in the file being reconstructed once no more blocks are found and can be parallelised for improved efficiency (Memon & Pal, 2006).

As with file structure based algorithms, its ability to reconstruct files nested within eachother is not documented, and would require further testing to ensure false positive, false negative, and corrupted results are kept to a minimum when implemented into OpenForensics.

**3. METHOD**

**3.1 Research**

At the time of writing, there are no open-source file validation techniques written in C#. Therefore, research into the inner workings of established file-validation and detection techniques and an understanding of said techniques will be pivotal for the success of the project.

Resources such as the source code of the file-carving software Foremost (Kendall & Kornblum, 2007) and Dr. Ethan Bayne’s original documentation for OpenForensics will be heavily utilized and built from, particularly during the prototyping phase of the project. This will provide a strong baseline for the main refactoring stage of development.

Current applications of file manipulation within DF will also be researched, including the basic form present in OpenForensics. These findings will provide a benchmark for the performance, and robustness expected in the evaluation of the final product.

Prior experience with the C++ programming language provides a solid foundation for learning to develop in C#, however, an understanding of the nuances between these languages and their capabilities will require research into their differing syntax and libraries. This will be done making use of theoretical and practical exercises available online and in C# development books and exercises, first getting familiar with smaller code fragments before moving onto more complex prototypes.

As this program will be refactored with security in mind and released to the wider public to download for free, knowledge of how attackers could potentially exploit the written code is vital for the safety of the application’s users. The refactored file validation and detection code will be written following general security programming guidelines as described in Programming .NET Security: Writing Secure Applications Using C# or Visual Basic .NET 1st Edition (Freeman, 2003) and following freely available security programming advice from lectures, online tutorials, and trusted forums such as those available on GitHub.

**3.2 Development**

Development will be performed locally in a VS 2022 Environment due to its large library of extensions and ability to scale with larger projects. Its implementations of IntelliSense code-completion aid and in-depth debugging software will greatly improve workflow for the project.

The most recent version of OpenForensics (v1.85b) will then be downloaded and imported into the development environment, where most development will take place. Prototypes will be developed in VS 2022 to demonstrate and understand the advantages of different file detection methods, though this will be done as a separate project file from the OpenForensics source code. The scope of data analysed by the prototypes will range from reading in and reconstructing a simple .txt file to eventually being able to handle gigabytes worth of varied filetypes and data. For these more advanced builds, the aim will be to successfully process all the data from a sample forensics forensic disk image, filled with the sort of files and metadata expected to be found in a real forensic investigation.

An Agile development framework will be utilized throughout development. This methodology’s strengths are its consideration of careful planning, regular reviews, and scalability for teams both big and small, all of which fit the criteria for this development project. Prototypes will be sent to the supervisor on a bi-weekly basis as development progresses as an opportunity to gather feedback before the final submission, covering the “Review” section of this chosen framework.

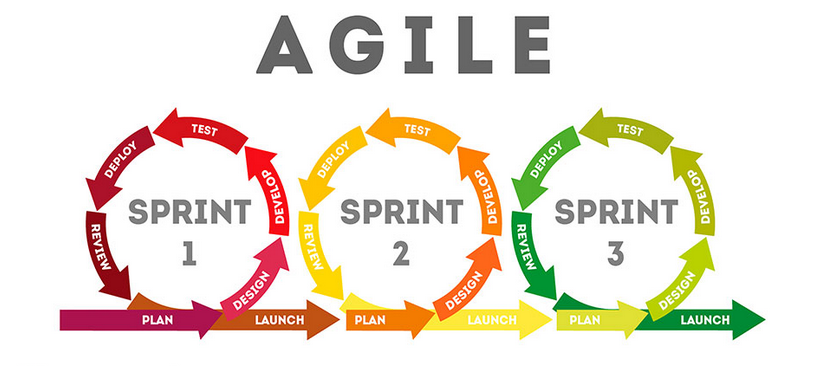


Figure 3 - The Agile development framework (ScrumStudy, 2018)

Any discovered security issues found within the newly implemented algorithms will be corrected as the project progresses and communicated to the OpenForensics author rather than being noted and worked on later in development. This is due to the importance of software security, particularly since security oversights can greatly affect the unknowing consumer’s experience and privacy depending on the severity of the issue. Security issues found in the legacy code during evaluation or testing will also be communicated to the supervisor, though remediating those issues is not within the scope of the project.

Investigating the security of the written code throughout development will cover the “Testing” and “Analysis” phases of Agile development from which subsequent Sprints can be built upon, restarting the cycle with updated plans and aims.

**3.3 Evaluation**

To evaluate the effectiveness of the implemented file detection functions, the C# “Stopwatch” class and the VS Performance Profiler will be utilized as a way of gathering performance and execution data. The class can measure the time it takes for specified code fragments to execute down to the nanosecond by adjusting its tick frequency, allowing for accurate and comparable performance measurements.

The program’s efficiency with memory space allocation will also be investigated through the use of a debugger to ensure the refactored program handles its provided space in memory correctly, clearing unused data when required and preventing memory leaks from taking place.

The gathered performance data will then be written into a Microsoft Excel spreadsheet, from which multiple charts can be created comparing the original and newly implemented code’s performance. These graphs will be analysed and discussed within the dissertation throughout the development cycle to show progress on the performance of the implemented code and determine areas of improvement.

**4. Summary**

In summary, this project will investigate the current implementation of file-detection and validation techniques present in OpenForensics and document the refactoring process, altering the present code to analyse input files and output results in a more effective and reliable manner. The aim of this investigation is to make the program practical in formal forensic investigations.

The outcome of this project is expected to be a VS 2022 Solution written in C# featuring the newly implemented algorithms, capable of processing and presenting results with improved performance.

A dissertation format will be used to document the development of these improved processing algorithms and provide insight into the methodologies behind producing an effective file-detection and validation algorithm, as well as a critical analysis and benchmark against currently implemented systems.

**5. REFERENCES**

Alsop, T., 2019. *Share of households with a computer at home worldwide from 2005 to 2019.* [Online]   
Available at: https://www.statista.com/statistics/748551/worldwide-households-with-computer/

Bayne, E., Ferguson, I. & Sampson, A., 2018. OpenForensics: A digital forensics GPU pattern matching approach for the 21st century. *Digital Investigation,* pp. S29-S37.

Davidson, D., Berry, M., Rathbone, D. & Burnett-Stuart, C., 2019. *Forensic science and the criminal justice system: a blueprint for change,* London: House of Lords.

Freeman, A., 2003. *Programming .NET Security: Writing Secure Applications Using C# or Visual Basic .NET 1st Edition.* Sebastopol, California: O'Reilly Media.

HMIC, 2015. *Online and on the edge: Real risks in a virtual world,* London: HMICFRS.

Kendall, K. & Kornblum, J., 2007. *ForeMost.* [Online]   
Available at: https://foremost.sourceforge.net/

Memon, N. & Pal, A., 2006. *Automated reassembly of file fragmented images using greedy algorithms,* Piscataway: IEEE.

ScrumStudy, 2018. *Blending Agile Frameworks for Project Success.* [Online]   
Available at: https://blog.scrumstudy.com/blending-agile-frameworks-for-project-success/

Warlock, 2018. *File carving.* [Online]   
Available at: https://resources.infosecinstitute.com/topic/file-carving/