

Introduction

Our main goal of this project was to create an awesome tool that was very useful in investigations and data forensics. We wanted to build a working file carving tool that could find and recover deleted JPEG images from a raw disk image. The reason we did this was to tackle a very frequent and frustrating problem in digital forensics, which is recovering evidence when it has been “deleted” from the file system. What our file carving tool does is it searches the drives unallocated space to identify the images by their unique signatures. This makes it a crucial utility for investigations where picture evidence may have been deleted by the suspect.

Technical Implementation

The foundation of the project is the `jpeg_carver.py` script, which we developed using Python 3. We chose Python because it's familiar, easy development, and is good with working with binary data, which is essential for low-level disk analysis. Since we had to access a physical drive directly like `PhysicalDrive0`, we gave specific instructions that made it clear the tool needed to be run using VS Code as Administrator to avoid any annoying permission errors, which is also pretty standard forensic practice. The tool uses a very targeted signature-based carving method. We hardcoded the binary signatures for JPEG files: `b"\xFF\xD8\xFF"` as the start signature (`JPEG_START`) and `b"\xFF\xD9"` as the end signature (`JPEG_END`). The main function, `carve_jpegs`, reads the disk image in large chunks (default 4MB blocks) at a time, which catches signatures that might split across a block boundary. Once the start signature is found, the data gets streamed into a buffer until the end signature is hit, treating the bytes between these two points as a candidate image. We also added some minimal heuristic checks in the `looks_like_jpeg` function. This check ensures the file is within defined size limits and looks for key metadata markers like JFIF or Exif near the header, which helps filter out corrupted files that just happen to match the start/end bytes. We relied on standard Python libraries, mainly `argparse` to handle command-line inputs and `os/sys` for file system operations and error handling. This allowed us to control the carving process using specific parameters like setting a file limit (`-n 50`), minimum size (`--min-size 20000`), and maximum size (`--max-size 5000000`) as demonstrated in our test command2: `python jpeg_carver.py "\\.\PhysicalDrive0" -o Recovered_JPEGs -n 50 --min-size 20000 --max-size 5000000`.

Results

We tested the tool in a realistic scenario: deleting known images from a live drive and then running the carver on a clone of that drive. The test outcome was a successful execution that recovered 3 image files from the drive. The tool generates two main types of output: a console log that displays real-time status, including the size and validity check results for each file candidate, and an output folder (`Recovered_JPEGs`) containing recovered images. The console output would include confirmations like: `[+] VALID JPEG saved: recovered_0001.jpg (1.2 MB)`, demonstrating that the tool successfully matched the signatures, passed the validation checks,

and wrote the data to a new file. The successful recovery of the three test images validated that the core signature-matching logic works on real-world deleted data.

Lessons Learned & Conclusion

The simple header/footer matching was very effective for recovering files that were not fragmented and had intact signatures, which was a success. Additionally, including the command-line parameters for size filtering (`--min-size` and `--max-size`) was also beneficial for controlling the output and stopping the tool from wasting time on massive, corrupted chunks of data. However, the biggest challenge was that the tool worked intermittently, which is a known limitation of strict signature carving. When the images were we assume fragmented across the disk, the tool we believe would often captured the correct parts but then would maybe read past the other parts of the picture, resulting in a corrupted, oversized, or invalid file recovery. Furthermore, the tool's specialized nature meant it was restricted solely to JPEGs.

Conclusion

The project successfully delivered a functional, foundational signature-based JPEG carver, validating the core principles of low-level data recovery. If we were to rebuild this project, we would make one major change, which would be to include other file formats and make the tool more well rounded. But for the purpose of the project we think this was a good scope to have. We would expand the format support beyond just JPEGs by integrating new logic for common file types (like PNG or GIF), allowing the tool to be a more versatile utility in digital forensic investigations.