Search and CompAre Reverse (SCAR): A Bioinformatics–Inspired Methodology for Detecting File Remnants in Digital Forensics

George Grispos, William Mahoney, Sayonnha Mandal
School of Interdisciplinary Informatics, University of Nebraska-Omaha, USA
ggrispos@unomaha.edu
wmahoney@unomaha.edu
smandal@unomaha.edu

Abstract: A storage device may contain data that an individual is legally or morally not allowed to possess. Or, a disgruntled company employee may intentionally destroy corporate files, assuming once deleted the information is lost forever. The data could take the form of a database owned by a competitor, illegal images, or videos, or trade secrets or confidential business information. Fragments of the data may very well still be present on the disk drive, for example, and forensic tools may be capable of recovering some of the confidential information. This paper introduces Search and CompAre Reverse (SCAR), inspired from tools used in the bioinformatics community. The contribution is an initial empirical investigation into the use of this bioinformatics-inspired approach to deduce the partial existence of patterns in cases where traditional digital forensics tools cannot detect the type of the file due to overwriting the file signature portion of the file.

Keywords: Digital Forensics, Digital Investigations, Filesystems

1. Introduction

In the last decade, the digital forensics community has witnessed an increasing number and volume of storage devices being seized and examined by forensic investigators, which has created large and lengthy backlogs of investigations (Quick and Choo 2014). One of the reasons for these backlogs is the increased processing times needed to forensically image and analyse the storage devices under investigation. As a result, a growing number of digital forensics researchers (Richard III and Roussev 2006, Garfinkel 2010) have proposed that the community develop tools and techniques to decrease processing times needed to analyse individual devices.

Many current digital forensics tools have been designed to decode and present the underlying filesystem under investigation. For example, these tools can help recover files that have been marked as 'deleted', and then attempt to recover these files and present the results to the investigator. This same approach is used for both partially and 'totally' recovered files. Hence, these tools can be used by an investigator to provide evidence that a file or directory was once present on a storage device (e.g., hard disk drive), but has since been removed or deleted. However, a disadvantage to this approach is that in some cases when a file or directory is deleted, the 'space' occupied by this filesystem object could be reclaimed by the filesystem to store new files. The problem arises that many forensics tools require the detection and recovery of the file signature part of the file, to decode and present the file correctly to the investigator. Should the filesystem reclaim the signature portion of the file, these tools will be unable to decode the file signature and the remaining parts of the file could be marked as slack and grouped together with other slack files. Examining slack files can be time consuming, with an investigator effectively looking for a 'needle in the haystack'.

This paper attempts to solve this problem by introducing an alternative forensics methodology called *Search and CompAre Reverse (SCAR)*, inspired from previous research in the bioinformatics community. At a high-level, SCAR provides an investigator with a visual indication that a certain pattern (e.g., a file) either exists or once existed within a forensic image created from a storage device under investigation. The research contribution is an initial empirical investigation into the use of this approach to deduce the partial existence of patterns in cases where traditional digital forensics tools cannot detect the type of the file due to overwriting the file signature portion of the file. The SCAR approach can be used to determine if 'fragments' of a known file can be found on the blocks of a storage device under investigation, without relying on the underlying filesystem. The rest of this paper is structured as follows. Section two motivates the research, Section three describes the SCAR methodology, Section four presents the results of our initial analysis, and Section five provides results and their potential impact on the digital forensics community. Section six concludes the paper and presents ideas for future research.

2. Motivation

In short, digital forensics is the process of preserving and analysing digital evidence (Palmer 2001). A variety of digital forensics tools have been developed to assist investigators analyse storage devices associated with a specific digital crime or incident (Grispos and Bastola 2020, Freyhof et al. 2021, Grispos and Mahoney 2021). Many of these tools work by providing an investigator with a layer of abstraction between the bits stored on a storage device (e.g., a hard disk drive), and how this data is actually stored and organized on the storage device itself (Roussev et al. 2013). Common forensics tools used for the analysis of storage devices include Forensic Toolkit (FTK), EnCase, and Sleuth Toolkit.

These tools can be used to examine a specific filesystem and detect deleted files and directories, which is accomplished by the tool 'decoding' the filesystem details, and then searching for the specific file entries which are marked as deleted, but not yet overwritten. In some cases, data within the file (e.g., file signatures) may indicate the type of file (e.g., JPEG file). However, one issue with this approach is that it only works if the original filename entry and other file metadata can be recovered from the filesystem. This raises two questions: *what if the entry for a deleted file cannot be recovered? It is still possible to determine whether a certain file was stored on the filesystem, if we know what artifact we are looking for during our investigation?* While this answer can be found using many of the above tools, it is both cumbersome and time consuming.

While traditional forensics tools can be used to detect and recover deleted files, they usually require that the file header can be detected and decoded. Otherwise, the deleted file cannot be decoded or digested by these high-level abstraction tools. In cases where the file header has been overwritten, but the data segment of a deleted file is available, this data is often clustered together by the tools as 'slack files' (Casey 2009). These slack files can be very large in size and consist of unstructured data that have been recovered from 'orphaned' data segments that do not include file header information. Hence, the aim of this research is to develop a methodology that can be used to determine if the fragments of files, which could still exist in the filesystem of a storage device, can be used to determine, and identify known files. The nomenclature used here is to search for patterns within filesystem images; a pattern is simply the term used for a binary artifact such as a JPEG file. It must be noted that the actual filesystem type is not a factor in our approach.

A pattern can consist of an entire file, or a pattern can be a portion of a file. For example, if an investigator suspects that a filesystem contains an illegal JPEG picture, the investigator might use only a fraction of the JPEG picture as a pattern; in this manner the investigator does not need to have the original picture but can still decide as to whether that portion of the image is present on a filesystem. The aim of SCAR is to perform partial matches of blocks of bytes within the patterns against blocks of bytes within the filesystem (in this example, a JPEG picture within a filesystem). Matching partial blocks is critical because the patterns may be only partially resident on an image; the pattern or file may have been deleted and subsequently overwritten by the contents of new files.

This partial overwriting of blocks is quite common. Files are stored in blocks on the device, assigned by the operating system, and tracked as part of the filesystem. When a file is deleted, the data portion of the file is not overwritten; simply the link from the name of the file to the data for the file is removed, and the blocks allocated to the file are marked as 'available'. As files are created, these free blocks will eventually contain new data. Deleting a file and never creating any new files will leave the data intact for the life of the device. But if new data is created on the device, these blocks containing the previous information will gradually be eliminated. Moreover, if newer files are smaller than the original block size, older data can remain in the original block allocations. Hence, SCAR scans blocks from the back towards the front in order to identify older deleted files in larger blocks.

3. Approach

SCAR is influenced by previous work from the bioinformatics community since the problem being addressed could be very similar. While bioinformatics software is used to scan for the presence of a particular gene in a sequence of DNA, a forensics investigator might design software to scan for the presence of a file in a storage device.

3.1 Prior Bioinformatics-Inspired Cybersecurity Efforts

In the past, cybersecurity researchers have attempted to leverage the similarities in searching for a specific gene in a digitised sequence of DNA, with searching for digital artifacts in filesystems. Much of this research has expanded the ideas and approaches proposed by Altschul et al (1990) for the Basic Local Alignment Search Tool

(BLAST), a computer algorithm that can be used online (National Center for Biotechnology Information n.d.), or installed on a local machine. BLAST is used to rapidly align and compare a query DNA sequence with a database of other, longer sequences, to find 'alignments' (i.e., approximate matches) that can be used a starting point for further investigation.

Within cybersecurity, BLAST has been used to classify new malware (Pedersen et al. 2012, Pedersen et al. 2013), where it was used to search for similarities between sequences of a digital artifact (a virus), and a set of sequences representing other digital artifacts, which are stored in a BLAST database. Computer virus code was converted into the representation necessary for the tool (e.g., search strings), which were then compared using BLAST to previous collections of computer viruses, to classify the new malware.

Fink and Oehmen (2012) developed an approach that extends the Machine Learning String Tools for Operational and Network Security (MLSTONES) project active at Pacific Northwest National Labs. They demonstrated that their approach could be used to convert digital binary files into amino acid sequences, which are then matched against the MLSTONES system.

The CodeDNA project (Glendenning n.d.) uses a longer alphabet to represent executable code. Sections of software are assigned a 'letter' depending on the function, and the functions are higher-level. One code might indicate a file operation, while another indicates creating a socket connection. Once potential malware is analysed and then translated into a sequence, the CodeDNA software can then be used to compare the sequence to the DNA of known malware to identify resemblances.

Other uses for DNA analysis in cybersecurity include encryption (Priya and Saritha 2017), using sequence alignment for network threat recognition (Kozakiewicz et al. 2007), and using pair-wise sequence alignment to test for masquerading (Coull et al. 2003). Hence, while previous research has examined the use of BLAST for various applications in the cybersecurity domain, this tool could also, in theory, be used to scan an entire filesystem for remnants of known files for use within digital forensics. However, one challenge that could emerge is the that one must convert the filesystem and the known file pattern into DNA sequences in ASCII, and only then provide these DNA sequences to the tool.

3.2 Description of SCAR

Our SCAR methodology is an expansion of concepts presented by Neamatollahi et al. (2020). The approach presented designed towards an exact match of a subsequence within a larger DNA database. The authors present a set of algorithms initially scan the DNA database to create 'windows' where there is a potential match with the search sequence. These 'windows' represent starting points for subsequence matches, which are then examined to a greater extent for the exact matching.

One of the algorithms presented by Neamatollahi et. al. is the Processor-Aware Pattern Matching (PAPM) algorithm. This algorithm is considered processor-aware, since rather than examining and comparing DNA byte by byte, it compares several bytes at once; a 64-bit CPU will examine eight ASCII characters. The algorithm, unfortunately, assumes that the CPU allows unaligned 64-bit memory accesses, so while this would execute on x86-64, it will not work on many reduced instruction set architectures.

In the case of digital forensics, PAPM is a close fit, but with the allowance that the search proceeds in opposite direction. The aim in this case is to determine whether file pattern fragments still exist on the filesystem. These pattern fragments may be partially overwritten in the disk block, and if so, the section of the disk block that would have been overwritten is at the start of the block, with the original fragment located at the back. Thus, a portion of the SCAR algorithm is to search the disk blocks in the same manner as PAPM but from the end of the block towards the front. There is only one starting point window to consider, and that is the last CPU word in the block under test. Therefore, the search comparison proceeds from the last CPU 'word' of the block towards the front of the block, and if a word-size mismatch occurs, the last trailing bytes are tested so that an exact number can be returned. SCAR can simplify DNA subsequence searching, as we are only interested in exact matches on partial blocks, as opposed to bioinformatics searches, where the subsequence may have minor differences.

3.3 SCAR Algorithm

Algorithm 1 provides a broad overview of the methodology used by the approach. The algorithm scans for file contents (patterns) within an acquired forensic image. In Algorithm 1, "image blocks" are large sections of a filesystem, while "pattern blocks" are smaller sections of what is currently being searched for. The algorithm

maintains a “search set” which is the information necessary to look for one file within the image blocks. Search set entries are treated as a pool of resources and are assigned to pattern files as needed. Each search set entry has an indicator which tracks the status of the entry, whether the entry needs the next block of pattern data, whether it is ready for using the CPU, and so on. All search set entries have a status of “available” when the algorithm starts.

```
while (keep_going)
       Reset the image file input to the beginning of file

       while (keep going and more image blocks to read)

              for (each entry in the search set)
                     if (this entry is “completed”)
                            Print the results
                            Mark the entry as “available”

              for (each entry in the search set)
                     if (this entry is “available”)
                            Open the next pattern and assign to this entry
                            Mark the entry as “needs_data”

              for (each entry in the search set)
                     if (this entry is “needs_data”)
                            Read the next block from the pattern
                            Mark the entry as “needs_CPU”

              for (each entry in the search set)
                     if (this entry is “needs_cpu”)
                            Perform the PAPM right-to-left search

              for (each entry in the search set)
                     if (this entry is “needs_cpu”)
                            This entry has scanned all blocks in the image
                            Mark the entry as “needs_data”

              for (each entry in the search set)
                     if (all entries are “available”)
                            We have processed all patterns
                            keep_going = false
```

Algorithm 1: Primary SCAR Program

The motivation for this algorithm is the consideration that, potentially, an investigator could be searching for different patterns when searching a storage device, and that these patterns are not of a uniform size. As a result, smaller pattern files will result in a quicker search as compared to larger pattern files, making entries in the ‘search set’. available for use by the next pattern file. Further, the program is trivially parallelisable. This is an important requirement as one assumes that the input (a forensic image) is likely to be the size of a filesystem. The overall algorithm is designed to take advantage of the POSIX thread library available on Linux and other systems. Tuning the size of the search set to the number of CPU cores available maximizes the utilization of the processors; looking for ten patterns at a time would cost little more in terms of time than looking for five, assuming there are sufficient CPUs. Moreover, after the threads are started and performing their right-to-left search of the pattern blocks, the main loop of the program initiates a read for the next block of the image. Observation shows that in general this allows the next image block to be available by the time the threads have completed the search through the current dataset, thus overlapping the I/O operation.

The overall flow of the method is to: a) create as many search set entries as necessary to match the CPU cores, b) load in as many patterns as there are search set entries, and c) for each entry compare the pattern blocks against the current image block, for all pattern and image blocks currently available. This is repeated until the entire forensic image is scanned, while filling in search set entries with new patterns as the entries become available. It is necessary to have larger patterns scan through the image more than one time, and this is time consuming. From an optimization perspective, if a pattern (e.g., a JPEG file) has been completely matched, there is no need to continue searching for the pattern blocks. Thus, a pattern that is deleted from the filesystem, but not overwritten can cause the search to finish for that pattern.

```
// t is the address in memory of the image block,
// p is the address in memory of the pattern block
match_count = 0
t_ptr = t + size of the image block - size of a word
p_ptr = p + sized of the pattern block – size of a word
while (t_ptr > t and p_ptr > p and word at p_ptr == word at t_ptr)
      match_count = match_count + size of a word
      t_ptr = t_ptr – size of a word
      p_ptr = p_ptr – size of a word
```

Algorithm 2: Processor Aware Pattern Matching

What remains is the description of the PAPM right-to-left-search, based on ideas proposed by Neamatollahi et. al., (2020) and shown above in Algorithm 2. This algorithm progresses right to left within the block, and counts matches in terms of 8-byte (64-bit) quantities instead of bytes.

3.4 Performance Parameters

As this is an experimental algorithm, the parameters for the algorithm should be changeable. In terms of performance, the algorithm is influenced by several different factors:

- **The size of the filesystem image block currently in memory**: There will be a certain amount of time required to scan for the pattern blocks within the image blocks, and larger image blocks will of course take proportionally longer to scan. But the trade-off is that it is possibly faster to read larger amounts of the image into memory in one input request to the operating system.
- **The size of the pattern block currently in memory**: The same question exists for the pattern blocks. Suppose one reads in 64KB of a JPEG file, but that the file is 65KB in size. With the current algorithm, the remaining 1KB of data will need to be compared against the entire filesystem, necessitating a complete pass through the filesystem for only a small portion of the pattern. Here one can assume that 'bigger is better' and for this reason allocating 128KB would perform better. But it is not essential to have the entire original file as the pattern; one could take a 64KB sample from the middle (aligned on a block) of the file and use this as the pattern. The files used in the evaluation were all less than 64KB in size, so in terms of performance this parameter was not tested.
- **The number of CPU cores available for searching**: Here the trade-off is that if the hardware has 16 cores, one can perform 16 pattern searches simultaneously. The algorithm operates correctly whether the number of threads is smaller than, equal to, or larger than the number of cores. The trade-off is the time required to start the threads and to rendezvous with their completion.

It is necessary to select parameters for SCAR that are appropriate for the user's hardware. For example, it may not make sense to execute 24 threads on a machine with two cores. To evaluate the performance of the algorithm, the authors explored some of the above issues, using a HP ProLiant with 160 GB and 24 cores. The following results were observed. Adding more threads improves performance, but only up to a certain point. Various Linux tools such as *htop* and *strace* show that most pattern-block-to-image-block matching fail quickly; the overhead of starting threads, which then fail, tends to consume more time than the payback of testing the patterns in parallel. A thread limit between 4 and 10 seems appropriate on this hardware. This leads to the question: is it better to read smaller or larger blocks of the patterns and the image? To determine this impact, the size of the in-memory block from the filesystem image was varied, since it is the larger of the two variables. On the HP hardware, the time required to read filesystem images in blocks over 8MB will have an impact on the total elapsed time required to perform the tests. The threads running the PAPM test will complete and then wait before the next filesystem image block is available. Using the Linux command *htop*, it quickly becomes apparent that a very large setting for the size of the image blocks means that the available CPUs are not being utilized 100%. Rather, threads start and immediately complete, and the algorithm waits for the next block.

It is advantageous to run the algorithm with a few samples to determine good settings for specific a hardware setup. SCAR allows a user to set from the command line, the number of threads, the size of each pattern block, and the size of the image blocks, as well as the file containing the image and the directory containing the patterns.

4. Forensic Usability of SCAR

SCAR was evaluated to determine the usability and effectiveness of using the approach to locate patterns (files) in a filesystem, during a forensic investigation. The test environment consisted of a virtual machine running

Ubuntu Linux version 20, setup with four partitions. The primary partition was used to store the operating system and to execute SCAR, while the three other partitions (hereafter referred to as the 'test partitions') consisted of a FAT32 filesystem, approximately 64MB in size. The test partitions were filled with JPEG image files ('the test dataset') from data sets of images of dogs (Khosla et al. 2011) and flowers (Nilsback and Zisserman 2008), in order to represent 'pattern files. These JPEG images were selected at random, and the same random selection were then copied onto both test partitions. In cases where random parts of a pattern file are overwritten, the Linux device */dev/urandom* was used as the source of the random bytes. The test dataset on the test partitions were then manipulated in three different tests as discussed below. At a high-level, these manipulations involved a treatment (deletion and/or overwriting of files) being applied to the test dataset, and then the analysis of the test partitions using SCAR methodology and Autopsy, a toolkit commonly used by the digital forensics community. Ten files were randomly selected from the test dataset to be used as patterns for SCAR searches in all three tests.

4.1 Detection of Deleted Files

The first test involved the detection of deleted files within the first test partition. This involves checking that the algorithm operates correctly, and that Autopsy detects the deleted files as a confirmation. After mounting the test partition, ten random files are deleted from the filesystem. The filesystem is then unmounted and subjected to both the SCAR methodology and Autopsy. With respect to SCAR, the algorithm detects 100% of the pattern blocks for all ten files. Likewise, when the partition was examined using Autopsy, all ten files were recovered and presented for analysis. In both cases, the results are not unexpected since the contents of the files have not been destroyed or overwritten. Hence, both SCAR and Autopsy can 'see' the deleted files.

4.2 Deleted Files with Random Blocks Partially Overwritten

The purpose of second test was to investigate if SCAR can be used to detect partially deleted files from the test partitions. For this test it is assumed that the blocks, which are no longer part of file, are assigned in some order, either by disk cylinder or by a closest-seek-time-next algorithm that is not related to the ordering of the blocks in the file. In these cases, blocks of the file will be destroyed in random order. Using the second test partition, ten random files were deleted from the filesystem: 25% of the blocks in each file were selected at random, and of those blocks, the first 25% of the block was overwritten with random data. It can be assumed that for some of these files, the file headers could still be present on the filesystem, with others may have been destroyed. This sets a scenario where files are deleted, and the filesystem reallocates the space to 'new' files. The analysis using Autopsy revealed that all ten files could be detected, and an attempt was made to recover the data within these files. However, only one of these ten files could be visualised using the file viewer in the toolkit. Further analysis of this file revealed that it was the only file whose file header was still intact, allowing Autopsy to visualise the file to the user. Figure 1 below presents the original file from the test data set, and the file recovered using Autopsy with 25% of the file randomly 'deleted'. The test was also repeated by overwriting 50% of the blocks in the ten random files, as well as 75% of the blocks in the same files. In both cases Autopsy did not detect any of the files.

Figure 1: Original JPEG and JPEG Image with 25% Data Destroyed

The test partition was then analysed using SCAR and visualisations were created to show the file patterns visible on the partition. For example, Figure 2 presents the pattern file for the JPEG image shown in Figure 1, with 25% random blocks destroyed, 50% destroyed, and 75% destroyed. To visualize the pattern data, we use a scale colour-coded by the percentage of the pattern block that was discovered intact on the partition. For this specific

pattern, where there happen to be 52 data blocks, the visualization has eight pattern blocks horizontally, so the last row is only partially filled.

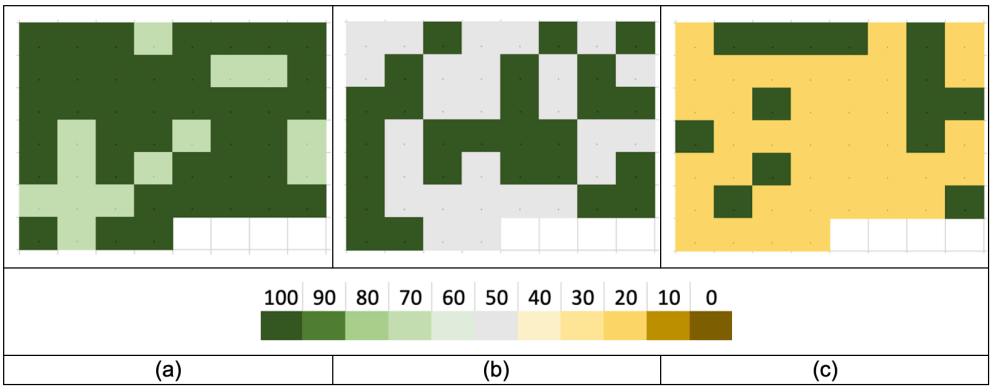

Figure 2: SCAR Detection with 25% Random Data Destroyed (a), 50% (b), and 75% (c)

The visualisations show that SCAR can still detect and confirm the remnants of the original file, even though the actual file is missing significant amounts of data. The SCAR output also shows why at 50% and 75% destruction the file cannot be rendered by Autopsy as the file headers are destroyed but can still be detected and confirmed by SCAR.

4.3 Deleted Files with Complete File Partially Overwritten

When a file is deleted, the blocks belonging to the file are available for other use, but different filesystems handle the free blocks in different ways. Blocks could be allocated in a first-in-first out order, or possibly selected by their proximity to other blocks in the same file. To create a realistic test scenario to mimic this situation, ten random files were chosen and 25% of the file content was overwritten with random data. The file was then deleted. These actions simulate the file being deleted and then the data blocks being reused in order. The test was repeated with 50% and 75% overwritten.

The analysis of the test partition with Autopsy revealed that the tool detects the deleted file contents, but the files are not recognizable as pictures. However, Autopsy only detects the file because the filesystem directory entry had not yet been re-used. If the entry is overwritten, Autopsy would not discover the existence of the pattern data on the filesystem image. As a result, this file could go undetected. The analysis of the third partition with SCAR (as shown in Figure 3) shows that in all three cases (25%, 50% and 75% overwritten) SCAR can still detect remnants of the file. The argument can then be made that that a percentage of the file pattern still exists on the filesystem, even if portions of the file have been overwritten and the entry in the directory for the file is destroyed.

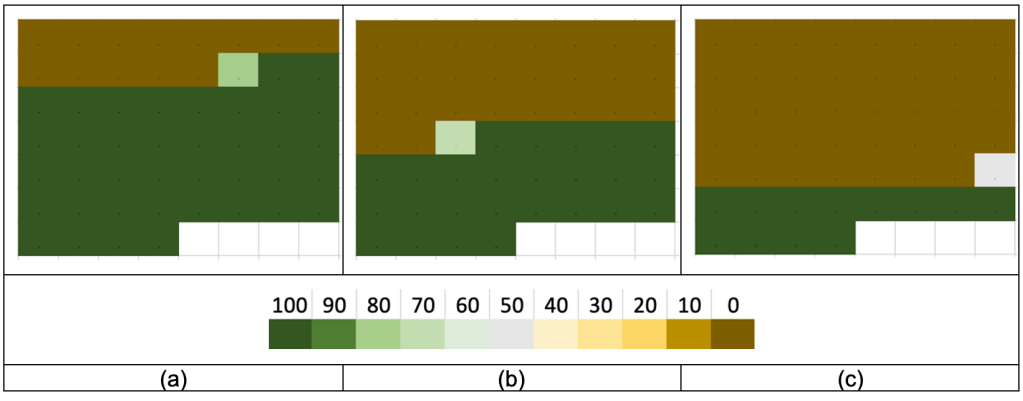

Figure 3: SCAR Detection with 25% Initial Data Destroyed (a), 50% (b), and 75% (c)

5. Discussion

A limitation to our approach is that the forensics investigator must have a copy of the original file to search for it in the investigation. Likewise, if the file in question changes frequently (e.g., a database) our approach might

be limited as the investigator may have a copy of the file which is outdated relative to what is on the media being investigated. While these limitations are evident, our approach also has advantages over other tools.

For example, forensic investigations of illegal images usually proceed by creating a hash value for the files on a suspect disk and comparing these hash values to those of known illegal images. Files which have been deleted, but can be recovered, can be compared as well. But simply changing one byte of an image file will change the resulting hash, and as a result the file may be present and the image still able to be viewed, but the file goes unnoticed by the investigator. Similarly, if portions of a file have been overwritten, a different hash comparison value will be presented to the investigator.

If the investigator has a complete copy of what they are looking for, the SCAR algorithm will search for the matching data. But there are understandable legal and ethical ramifications if the investigator has an exact copy of an illegal image. In contrast to simply having a hash of the file, or the entire file itself, the investigator could possess a portion of the data from original file; not enough for them to visualize the actual photo, but enough for them to conduct a search of the suspects disk. If there is a collection of several illegal files, the investigator could maintain a collection of portions of the files. For example, they could maintain a set of 32 Kbyte samples of each of the files, starting 4 Kbytes into the data. This skips past file header information and directly into the image, so it cannot be visualized on her computer. But the 32 Kbytes of file data can still be searched for on the suspects media. And since our algorithm operates on blocks of 512 bytes, any offset which is a multiple of 512 is acceptable. The advantage of SCAR, then, is in instances such as these; the hash method may not be applicable for one of several reasons, including a partially deleted file or a slightly modified file. But if a determination can be made that the data from the file is still present on the suspect filesystem, this still provides some evidence of the prior activity. Whether that evidence is 'sufficient' for a particular crime is considered out of scope for this research.

6. Conclusions and Future Work

This paper presents SCAR, a digital forensics methodology that is inspired from concepts initially introduced by the bioinformatics community. At a high-level, the purpose of SCAR is to scan storage media, looking for fragments of a known file that could exist in the filesystem of the storage device. This can be useful from an intelligence perspective; a business employee may have copied confidential corporate information that they should not have in their possession, or the employee may have deleted files intentionally through malice. Since many filesystems overwrite deleted files quickly after the user has deleted a particular file, SCAR can be used to search the filesystem of the storage media under investigation, providing information to the investigator if a portion of the file can still be found in the filesystem. The positive identification of the 'portion' of the file could provide an investigator with enough information to show a 'smoking gun' that the filesystem did at some point contain the known file. In this sense, SCAR could be particularly useful in the identification of information in investigations concerning illegal pictures and videos, corporate secrets, intellectual property, and other business assets, although we do not address the question of 'how much' evidence is 'enough' evidence.

There are several avenues for future work. First, we intend to refine the SCAR development and evaluate the methodology against a larger number of file types. While this initial research focused on the identification of JPEG files, future research intends to focus on other image and video file types, which are commonly sought in other digital forensic investigations. Second, future work will investigate the effectiveness and refinement of the SCAR methodology in other filesystems. The idea behind this avenue of research is to determine if SCAR can be successfully executed on filesystems such as NTFS, and what changes (if any) need to be implemented to allow the approach to be used in these filesystems. Finally, future research intends to examine the applicability of the SCAR methodology in RAID-based storage, commonly used in server-based storage solutions. Since RAID can 'split' a file over more than one physical storage device, this potential research will examine how SCAR can be modified to help a forensic investigator identify the existence of a particular file on physical storage devices used in a RAID configuration. As discussed in Section 3.4, there are several "tunable parameters" involved in optimizing the algorithm on a computer system, and future work could include an automated method for adjusting these parameters for optimal performance.

References

Altschul, S. F., W. Gish, W. Miller, E. W. Myers and D. J. Lipman (1990). Basic local alignment search tool. Journal of Molecular Biology 215(3): 403-410.

Casey, E. (2009). Handbook of Digital Forensics and Investigation, Academic Press.

Coull, S., J. Branch, B. Szymanski and E. Breimer (2003). Intrusion Detection: A Bioinformatics Approach. 19th Annual Computer Security Applications Conference, 2003. Proceedings., IEEE.

Fink, G. A. and C. S. Oehmen (2012). Final Report for Bio-inspired Approaches to Moving-Target Defense Srategies, Pacific Northwest National Lab.(PNNL), Richland, WA (United States).

Freyhof, M., Grispos, G., Pitla, S. and Stolle, C. (2022). Towards a Cybersecurity Testbed for Agricultural Vehicles and Environments. Proceedings of the Seventeenth Midwest Association for Information Systems Conference, Omaha, Nebraska.

Garfinkel, S. L. (2010). Digital Forensics Research: The Next 10 years. Digital Investigation 7: S64-S73.

Glendenning, M. (n.d.). CodeDNA: Scalable, High-Spoeed, High-Volume, Shareable Malware Detection. Cyber Security Division Transition to Practice Technology Guide, Department of Homeland Security Science and Technology Directorate: Cyber Security Division.

Grispos, G. and Bastola, K. (2020). Cyber Autopsies: The Integration of Digital Forensics Into Medical Contexts. 2020 IEEE 33rd International Symposium on Computer-Based Medical Systems (CBMS). IEEE.

Grispos, G., and Mahoney, W.R. (2022). Cyber Pirates Ahoy! An Analysis of Cybersecurity Challenges in the Shipping Industry. Journal of Information Warfare, Vol. 21 (3), pp. 59-73.

Khosla, A., N. Jayadevaprakash, B. Yao and F.-F. Li (2011). Novel Dataset for Fine-Grained Image Categorization: Stanford Dogs. Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC), Citeseer.

Kozakiewicz, A., A. Felkner, P. Kijowski and T. J. Kruk (2007). Application of Bioinformatics Methods to Recognition of Network Threats. Journal of Telecommunications and Information technology: 23-27.

National Center for Biotechnology Information. (n.d.). Basic Local Alignment Search Tool. from https://blast.ncbi.nlm.nih.gov/Blast.cgi.

Neamatollahi, P., M. Hadi and M. Naghibzadeh (2020). Simple and Efficient Pattern Matching Algorithms for Biological Sequences. IEEE Access 8: 23838-23846.

Nilsback, M.-E. and A. Zisserman (2008). Automated Flower Classification Over a Large Number of Classes. 2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing, IEEE.

Palmer, G. (2001). A Road Map for Digital Forensics Research-Report from the First Digital Forensics Research Workshop (DFRWS). Utica, New York.

Pedersen, J., D. Bastola, K. Dick, R. Gandhi and W. Mahoney (2012). Blast Your Way through Malware analysis Assisted by Bioinformatics tools. Proceedings of the International Conference on Security and Management (SAM).

Pedersen, J., D. Bastola, K. Dick, R. Gandhi and W. Mahoney (2013). Fingerprinting Malware using Bioinformatics Tools Building a Classifier for the Zeus Virus. The 2013 International Conference on Security & Management (SAM2013).

Priya, S. K. and S. Saritha (2017). A Robust Technique to Generate Unique Code DNA Sequence. 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), IEEE.

Quick, D. and K.-K. R. Choo (2014). Impacts of Increasing Volume of Digital Forensic Data: A Survey and Future Research Challenges. Digital Investigation 11(4): 273-294.

Richard III, G. G. and V. Roussev (2006). Next-Generation Digital Forensics. Communications of the ACM 49(2): 76-80.

Roussev, V., C. Quates and R. Martell (2013). Real-Time Digital Forensics and Triage. Digital Investigation 10(2): 158-167.