Sustainable Automated Data Recovery: A Research Roadmap

Jeroen van den Bos
Netherlands Forensic Institute
The Hague, The Netherlands
Zuyd University of Applied Sciences
Heerlen, The Netherlands
jeroen@infuse.org

ABSTRACT

Digital devices contain increasingly more data and applications. This means more data to handle and a larger amount of different types of traces to recover and consider in digital forensic investigations. Both present a challenge to data recovery approaches, requiring higher performance and increased flexibility.

In order to progress to a long-term sustainable approach to automated data recovery, this paper proposes a partitioning into manual, custom, formalized and self-improving approaches. These approaches are described along with research directions to consider: building universal abstractions, selecting appropriate techniques and developing user-friendly tools.

CCS CONCEPTS

• **Applied computing** → *Data recovery*; *Evidence collection, storage and analysis*; • **Software and its engineering** → *Automatic programming*; *Model-driven software engineering*;

KEYWORDS

data recovery, digital forensics, automated software engineering

ACM Reference format:
Jeroen van den Bos. 2017. Sustainable Automated Data Recovery: A Research Roadmap. In *Proceedings of 1st International Workshop on Software Engineering and Digital Forensics, Paderborn, Germany, September 4, 2017 (SERF '17),* 4 pages.
https://doi.org/10.1145/3121252.3121254

1 CHALLENGES IN DIGITAL FORENSICS

The amount of data stored on digital devices continues to increase as new applications arrive and existing applications are updated daily. As a result, digital devices take a more prominent place in our lives and are used for more and more activities. A large amount of business and personal information accumulates on these devices.

One effect is that digital devices are increasingly the focus of forensic investigations. This requires forensic investigators to be able to collect and analyze traces from many types of devices and applications. Requirements to answer questions about digital evidence within a very short timespan (e.g., 48 hours) are not uncommon.

SERF '17, September 4, 2017, Paderborn, Germany

ACM ISBN 978-1-4503-5156-0/17/09.
https://doi.org/10.1145/3121252.3121254

In this time a massive amount of data consisting of many different types of data needs to be investigated. Examples are communication (spread across e-mail, social networks, messengers, SMS, etc.), media (such as pictures, videos, audio), and applications (which create many types of documents). Even when there are no major complications (such as physical damage to the device or use of encryption), two major challenges are nearly always present: the size and variation in types of data.

1.1 The Data Recovery Phase

This paper focuses on the challenges around size and variation of data in a specific phase of the digital forensics process: recovery. While there are differences in terminology used to describe the phases, a high-level division into three main phases is widely agreed upon[3, 4]. This division separates the digital forensics process into *acquisition*, *recovery* and *analysis*, as shown in Figure 1.

Acquisition is concerned with making a secure copy of all the data stored on a device, so that it can be backed up and investigated without risk of data loss. Recovery handles converting the acquired data into information, i.e., turning raw data (bits and bytes) into interpretable information that humans can understand (messages, documents, pictures, etc.) Analysis refers to considering all recovered information and determining what is relevant to the investigation or what the answer is to a posed question.

This model can be extended in both directions (such as an *identification* and *reporting* phase before and after respectively) and all phases can be divided into multiple phases (such as *relating information* and *sorting* in the *analysis* phase). The model is kept small and simple because it is used here only to specify the meaning of the *recovery* phase as turning data into information. The rest of this paper will focus exclusively on the *recovery* phase.

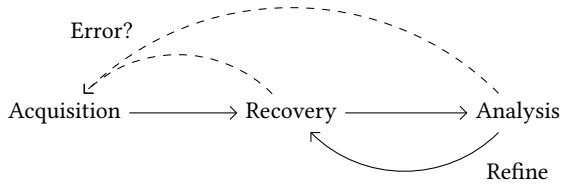

Figure 1: Basic model of a digital forensic investigation.

1.2 Size and Variation

Even though Moore's Law (predicting exponential growth in the capacity of digital devices) is always being said to be ending soon,

it appears to keep going indefinitely. Mobile phones, tablets, and USB drives are still regularly doubling in storage size and popular cloud services are offering *unlimited* plans at low prices.

While the amount of pictures and videos a person can produce in a day is limited, there are many developments that increase the amount of data that is produced. An example is that increasing bandwidth on wireless networks means live video streaming will become ubiquitous. Additionally, drones with cameras that can take pictures and film autonomously are gaining popularity.

Diversity of types of data is especially complex due to its many layers of variation. To illustrate this complexity, consider the most common type of information stored on any type of digital (storage) device: pictures. In an abstract sense, a picture is a very simple thing: a two-dimensional visual representation of something.

Practically however, there are different types of images, such as whether they are stored as a matrix of pixels (a bitmap-based image) or a collection of lines and curves (a vector-based image). Both types have many different ways to be stored on disk, such as JPEG, PNG and GIF for bitmap-based or SVG, EPS and AI for vector-based images. All these formats often have different standardized versions and variants, plus mandatory or optional extensions, such as JFIF (container) and EXIF (metadata) for JPEG. But even then, developers of digital cameras and image editing tools each tend to implement their own subjective interpretation of all these types, resulting in even more variations.

1.3 Automation and Evolution

The first challenge of size pushes forensic investigations towards *automated digital forensics*, the field of performing digital forensic tasks in software. This way, advances in performance and capacity of computers is not only a problem, but can also be leveraged to build ever faster and more scalable systems. However, the second challenge of variability means that this software must be extremely flexible and adaptable in order to implement and deal with all these variations. And not just during initial development: as applications and services are updated regularly, variations at all levels are discovered continually, prompting changes to the software.

For commercial software vendors that produce software applications not used in digital forensic investigations, the issues around variability are much smaller. Consider a word processing application such as Microsoft Word. While it is desirable from an interoperability point to view to have support for many file formats, a fixed set of supported formats tends to be acceptable to most users. Additionally, if a file of a supported type is opened by Word that has some unknown parts or implements some unknown variant, it is acceptable to simply inform the user that the file cannot be read.

In forensics however, it is generally unacceptable for some piece of evidence to not be considered because it does not conform to some standard or is not supported by some application. As a result, the burden of providing support for as many types of data as possible is firmly on the shoulders of the digital forensic investigator.

2 APPROACHES IN DATA RECOVERY

The process of recovering data is not straight-forward: the required techniques depend on the type of data that was acquired as well as what turns up in an initial recovery run. In the simplest case, all acquired data can be interpreted as being in some recognized and supported format and parsed accordingly. This is a fairly common case: network captures, live memory dumps, and copies of storage device contents can be interpreted and parsed as being sets of network packets, in-memory data structures and disk partitions with file systems respectively.

However, virtually every acquired piece of data usually contains some unrecognized parts, such as unknown protocols, memory data structures or file formats. Additionally, many formats such as memory layouts and file systems tend to contain areas that are supposed to be empty, but often contain remnants of previous entries, such as parts of the heap in memory that were deallocated or deleted files in a file system. In all those cases, manually investigating and reverse engineering the unknown parts to build a custom parser may eventually be required, but is infeasible under time pressure. Techniques used in these circumstances are *file carving*[10] and *entity extraction*[7].

File carving is based on the assumption that while some piece of data cannot be parsed in its entirety, it may contain parts that can. Effectively, file carving is the process of searching through unstructured data looking for parts that can be recognized. Techniques range from recognizing magic numbers in file headers and footers up to combinatorial approaches to reassemble deleted and fragmented files in a file system's unallocated space.

Entity extraction boils down to entirely the same thing, except that instead of searching and matching entire data structures or file formats, matching is done for small *entities* that can be useful regardless of where they occur. Examples are names, IP-addresses, phone numbers and bitcoin addresses. In this situation it does not matter whether the entities occur in some recognized or explainable location, but rather that they occur at all may provide some clue.

What this reveals is that all recovery activities depend on the availability of recognizers and parsers for as many different types of data as possible. Whether they are run directly on the data, used to validate assembled fragments by a file carver or used to scan over unknown data to locate entities, parsers make or break the success of data recovery. As discussed before, these parsers are preferably fast and scalable (in order to run quickly and on very large sets of data) and highly flexible and maintainable (because they must be continually adapted).

2.1 Types of Automated Data Recovery

This paper proposes that there are four different types of ways to perform data recovery, ordered by the level of automation involved. As data sizes as well as variation in encountered data types both keep growing, progressing to the highest level as described below will be required in all but the most trivial cases.

2.1.1 Manual Automated Data Recovery. In the digital domain, labeling an approach as *manual* is always relative: everything on or in relation to a digital system is generally performed using some form of software. In this case, manual refers to the fact that an investigation is performed by manually chaining different existing tools together. These tools can be custom developed tools or existing (open source) applications.

Manual data recovery usually boils down to attempting to mount a storage device using a file system driver in a virtual machine, or

loading a network capture into an analysis framework such as Wireshark. Whenever some known piece of data is then encountered (e.g., a picture), it can be examined using a standard picture viewer, trying different ones if the first is unable to display it. Regions of unknown data can be extracted and then carved using different file carvers, until some files are recovered. Possibly an investigator also uses low-level tools to search this data, such as grep or hex viewers.

This approach generally leads to the development of a set of standard operating procedures (SOPs) in order to prevent mistakes. While very resource-intensive, everything can generally be performed by skilled digital forensic investigators without the help of other professionals.

2.1.2 Custom Automated Data Recovery. When these SOPs lead to the development of scripts in order to automate them, the type of data recovery is characterized as *custom*. After scripting, adding custom tools to improve runtime performance and scalability as well as building libraries of parsers to handle types of data unsupported by existing tools is the next level in custom data recovery.

Achieving this level of automation typically requires software engineering skills to build and maintain the non-functional aspects of the solution. Additionally, reverse engineering skills are required in order to implement and maintain the libraries of parsers.

This approach leads to the development of an automated data recovery solution capable of providing levels of performance and coverage (i.e., support for as many types of data as possible) unavailable in off-the-shelf solutions. If the goal however is to increase coverage, the amount of effort on development and maintenance will quickly become extremely high.

The engineering challenges in custom automated data recovery are complex because they require optimization in different and opposing directions. Optimizations for runtime performance and scalability tend to be in direct opposition to those required to optimize for flexibility. For example, flexibility is generally realized by adding abstractions, while optimizing for performance is usually the result of merging or removing abstractions.

2.1.3 Formalized Automated Data Recovery. When a custom approach leads to the application of a language-based, generative or model-driven approach, it is described as *formalized*. What these approaches have in common is that they perform separation of concerns on the aspects that are difficult to consolidate in a solution built using scripting or general programming languages.

For example, at the Netherlands Forensic Institute (NFI) a domain-specific language (DSL) is employed to declaratively describe data structures such as protocols and file formats[2]. These descriptions are independent of concerns around runtime performance and also do not require the developer to have extensive knowledge of parser construction, since this is all handled by an interpreter.

This approach has many possible variants and extensions, including extracting the performance aspects into an abstraction as well. Given that flexibility and maintainability are the primary concerns facing investigators, abstracting this aspect is crucial, but others may be added. The advantage of these approaches is that the different aspects of the solution can be developed and maintained in complete isolation from the other aspects, reducing complexity and as a result, improving functional and non-functional qualities.

It is an open question however, whether highly productive tools such as model-driven engineering provide the necessary benefits to keep up with requirements. Software and service vendors such as Google and Microsoft are increasingly applying big data and artificial intelligence (AI)-based techniques to improve their software, which may result in refinements and modifications to the types of data they produce at an extremely high pace.

2.1.4 Self-Improving Automated Data Recovery. Just as the only feasible approach to dealing with the amount of data produced by automated solutions is to automate the forensic applications, there is a similar approach to handling big data and AI-approaches. Once an automated data recovery solution is capable of (partially) automating the process of adapting its support for types of data, it is considered to be *self-improving*.

There are many possible approaches in this area, but a key requirement in digital forensics is that the output must be verifiable and explainable by humans. If we base a self-improving approach on the tools of the formalized approach, we have a clear and usable *intermediate language* to target with any AI-technique.

Simply put, it does not matter how an abstract description of some data type is derived, as long as it can easily be understood and verified afterwards. This is in contrast to a solution where an entire recovery application, or even a parser written in a general programming language would be generated. In both those cases, the resulting application or code would most likely be very difficult to understand and thus verify. This is not a far-fetched assumption, given that engineers tend to have difficulty understanding code written by other engineers, let alone code generated by some genetic or evolutionary algorithm.

Plenty of approaches are imaginable in this space to assist in attaining a self-improving solution. An example is to take a binary data description language and use a genetic algorithm to discover a description for it given a set of input files. While discovering a meaningful description from scratch may be unfeasible currently, discovering variations of an existing description given a set of variant files could be achievable.

3 RESEARCH DIRECTIONS

To arrive at a fully automated, formalized and self-improving data recovery approach in digital forensics for which an actual implementation exists, many challenges remain in the approaches described in the previous section. Following is a discussion of the major issues, involving the formalized and self-improving approaches.

3.1 Metamodel Refinement

The formalized and self-improving approaches crucially depend upon an underlying model (i.e., metamodel) that allows expression of any possible data structure that may be encountered. While there is a large amount of research in parsing, most of it focuses on textual formats for use in compilers and other programming systems. Fortunately, there is some progress in the area of data description languages, from both programming languages (most notably the PADS and related projects[6, 8]) and security research.

Especially promising in this direction is *language-theoretic security*, an area that seeks to abolish the use of hand-crafted parsers in order to eliminate exploits caused by incorrect handling of inputs[9].

So while the motivation is different, the goal to find a universal non-executable description for handling any type of input data is the same. Most notable in this area are open-source libraries such as Hammer[13] (in C) and Nom[5] (in Rust).

A contribution by the NFI in this area is the development of the open-source METAL library (in Java)[1], which has the explicit goal of being capable of describing any (binary) data structure. To achieve this, it has support for pointers (i.e., non-local parsing), containers (i.e., parsing a previously parsed item or composition of items) and a rich expression system to define data dependencies.

The next step is to develop a library of diverse data structures using these systems, including network protocols, live memory layouts, file systems, containers and formats in order to evaluate and improve the quality of the underlying metamodels.

3.2 Synthesis Techniques

Perhaps because the previous direction has not yet been decisively resolved there is not much known about the most suitable algorithms to apply from the field of AI to the refinement, improvement and development of data descriptions. It is essentially a specific version of the *program synthesis* problem, which has been discussed and examined for over 40 years[12].

With the recent advancements in AI-technology and research, this field has been the subject of renewed interest, mostly in the area of *learning representations*. Progress is made to develop small human-readable programs to handle input-output problems based on examples[1, 11].

An often-discussed trade-off in this area is the size of the input and output data and the complexity of the programming language to generate code for. From this perspective, the automated data recovery problem is very specific: the input data can be huge (up to multiple terabytes at a time for network captures or storage device contents). At the same time, this explains the need to at least keep the complexity of the formalism to generate a representation in as low as possible, which is a clear goal of the data description parsing libraries discussed in the previous subsection.

Progress will depend mostly on assembling suitable sets of training data, such as a varied set of binary files of the same type which includes many variations. Such a set can then be used to experiment with and evaluate techniques in this specific version of the program synthesis problem.

3.3 User Interfaces

While the discussed parser toolkits make it much easier to develop parsers for binary data, they still have a long way to go to be truly user-friendly. Instead of embedded DSLs, which make up the libraries currently available, stand-alone languages can fully separate the concerns between tool development and data description.

A stand-alone data description language has the potential to connect the output of self-improving approaches to the need for human understanding. This requires that the language's metamodel is suitable for use with synthesis techniques and that the language itself is understandable for digital forensic investigators.

Along with the languages, user interfaces of investigation tools must be adapted as well. In order to understand the output of data recovery applications based on (generated) data descriptions, interactive environments are needed that visualize the provenance of any recovered piece of data. Even if some synthesized description ends up being usable but difficult to understand, its output can then be analyzed in context of the original input data and potentially clarified manually.

[1] https://github.com/parsingdata/metal

4 CONCLUSION

As Moore's Law keeps providing more computing power, bandwidth and storage keeps growing at a comparable pace. This means that digital devices become useful in more and more tasks every day and companies quickly provide applications for many different tasks. A result is that in digital forensics, investigators continually need updated methods and tools in order to deal both with more data and different types of data in an often very limited timeframe.

To structure research in the specific area of automated data recovery, this paper proposes partitioning the approaches into four different types. *Manual* refers to investigators using individual tools in some defined process. *Custom* refers to using software engineering to develop custom solutions. *Formalized* refers to abstracting over the custom approach by using model-driven techniques to separate concerns. *Self-improving* refers to automatically maintaining the formalized descriptions in order to eliminate the dependency on software and reverse engineers.

Three directions are most important to realize this final self-improving approach. First is improving formalized data recovery by refining and evaluating existing (meta)models in the area of binary data description. Second is the selection and evaluation of program synthesis techniques from the field of AI, as work in this specific area is currently non-existent in digital forensics. Finally, user-friendly languages and tools must be developed so that human investigators can understand and verify self-improving approaches.

REFERENCES

[1] Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. 2016. DeepCoder: Learning to Write Programs. *CoRR* abs/1611.01989 (2016).

[2] Jeroen van den Bos and Tijs van der Storm. 2011. Bringing Domain-Specific Languages to Digital Forensics. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*. ACM, 671–680.

[3] Brian Carrier. 2005. *File System Forensic Analysis*. Addison-Wesley.

[4] Eoghan Casey (Ed.). 2009. *Handbook of Digital Forensics and Investigation*. Academic Press.

[5] Geoffroy Couprie. 2015. Nom, A Byte oriented, streaming, Zero copy, Parser Combinators Library in Rust. In *Security and Privacy Workshops (SPW), 2015 IEEE*. IEEE, 142–148.

[6] Kathleen Fisher, Yitzhak Mandelbaum, and David Walker. 2010. The Next 700 Data Description Languages. *J. ACM* 57, 2 (2010), 10:1–10:51.

[7] Simson L. Garfinkel. 2013. Digital Media Triage with Bulk Data Analysis and bulk_extractor. *Computers & Security* 32, 0 (2013), 56–72.

[8] Trevor Jim, Yitzhak Mandelbaum, and David Walker. 2010. Semantics and Algorithms for Data-Dependent Grammars. In *ACM Sigplan Notices*, Vol. 45. ACM, 417–430.

[9] Falcon Momot, Sergey Bratus, Sven M. Hallberg, and Meredith L. Patterson. 2016. The Seven Turrets of Babel: A Taxonomy of LangSec Errors and How to Expunge Them. In *Proceedings of IEEE Cybersecurity Development (SecDev'16)*. IEEE, 45–52.

[10] Anindrabatha Pal and Nasir Memon. 2009. The Evolution of File Carving. *Signal Processing Magazine* 26, 2 (2009), 59–71.

[11] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. 2016. Neuro-Symbolic Program Synthesis. *CoRR* abs/1611.01855 (2016).

[12] Phillip D. Summers. 1977. A Methodology for LISP Program Construction from Examples. *Journal of the ACM (JACM)* 24, 1 (1977), 161–175.

[13] UpstandingHackers. 2012. Hammer, Parser combinators for binary formats in C. (2012). https://github.com/UpstandingHackers/hammer