DEX: Digital evidence provenance supporting reproducibility and comparison

Brian Neil Levine*, Marc Liberatore

Dept. of Computer Science, Univ. of Massachusetts, Amherst 01003, USA

ABSTRACT

Keywords:
Digital forensics
Digital evidence
Provenance
Forensic tools

The current standard and open formats for forensic data describe whole disk and memory image properties, but do not describe the products of detailed investigations. We propose a simple canonical description of digital evidence provenance that explicitly states the set of tools and transformations that led from acquired raw data to the resulting product. Our format, called Digital Evidence Exchange (DEX) is independent of the forensic tool that discovered the evidence, which has a number of advantages. Using a DEX description and the raw image file, evidence can be reproduced by other tools with the same functionality. Additionally, DEX descriptions can identify differences between two separate investigations of the same raw evidence. Finally, as a standard product of tools, DEX can allow quick fabrication of tool chains either as best-of-breed amalgams or for tool testing. We have implemented DEX as an open-source library.

© 2009 Digital Forensic Research workshop. Published by Elsevier Ltd. All rights reserved.

1. Introduction

Whether starting from digital data acquired from a desktop drive, mobile phone memory, or network trace, forensic examiners must ensure the integrity of the entire investigation. At the start, the authenticity and integrity of an acquired disk or memory *image* can be managed, along with information about data provenance and chain of custody, using one of several file formats (Garfinkel et al., 2006; Turner, 2005). However, the validation of parsing methodologies and the verification of the correctness of software tools and the evidence that such tools harvest are separate problems. Typically, forensic examinations have strong reliance on closed-source tools that have already been accepted by courts, and evidence is exchanged between parties as a prose description that makes reproduction time-consuming. Accordingly, the validity of an investigation almost always rests on the validity of the methods and the verification of the specific tools that are used. Moreover, the combined use of a series of tools is almost never verified as correct.

For example, a third-party examiner may have no precise documentation of reported evidence other than the extracted file and a statement such as "the image file was found in the Firefox browser cache by Encase". Search terms from browsers or slack data from files are harder to present, and often are displayed as a tool screen shot or hex dump. While these displays are necessary for the easier comprehension by non-experts of what was found, they are not sufficient for ensuring the integrity of the reasoning and process that harvested the evidence.

For some, only a review or static analysis of source code can determine if a tool is performing operations correctly in all cases (Battistoni, 2004; Carrier, 2002; Kenneally, 2001). Unfortunately, the tools that are under the most vigorous development, in the most popular use, and providing the greatest amount of support to non-expert users are closed source. Whether closed or open source, software testing and reliability is a challenging issue, and here we distinguish several different goals. *Validation* is the process of checking a specification for faults (Avižeienis et al., 2004). In our domain, the

---

specification is a set of properties for interpreting data as evidence. *Verification* is the process of checking a particular system (Avižeienis et al., 2004). By system, we most commonly mean software tools that implement a specification. For example, Carrier's book (Carrier, 2005) may be viewed as a specification, whereas The Sleuth Kit is a system.

In this paper, our goal is to improve the reproducibility and comparison of digital forensic evidence. We propose a simple canonical description of digital evidence *provenance* that explicitly states the set of tools and transformations that led from acquired raw data to the resulting product. Our format, called *Digital Evidence Exchange* (DEX) is independent of the forensic tool that discovered the evidence. A DEX description of evidence is sufficient for a third party using an independently developed tool to quickly extract the same evidence and verify that the reproduction is correct according to a known specification. Our approach and the use of DEX has a number of advantages.

Firstly, two independent examiners can exchange, compare, and reproduce the results of investigations in a precise fashion that is independent of the tools used. Rather than agreeing upon the tools used, they need only compare a DEX description of evidence. DEX separates evidence descriptions into layers that reflect the systems on which digital evidence reside. All computer systems are layered, and mirroring the layers in DEX as a tree is not difficult. Accordingly, a comparison of two DEX trees reveals differences in how evidence was interpreted from the source image and through all system layers in between.

Secondly, using DEX's standardized outputs, a combinatorial number of tool chains can be used to verify a specific piece of evidence or to validate against a known test case. For a chain of *x* tools, with *y* alternate tools at each step, we have y^x outputs that can be compared. This process is often called *N-version programming* (NVP) (Avižeienis, 1985) and is a well-known method of increasing software reliability via redundancy. NVP can be used to discover when two tool chains produce different results; implementation faults that are common to both tool chains, even if independently developed, would not be illuminated (Knight and Leveson, 1986). Without DEX, NVP would be a cumbersome, manual process. Notably, NVP with DEX works in principle with closed- and open-source tools.

Thirdly, by agreeing on a common output format, a series of small tools can be combined together in a best-of-breed amalgam, potentially forming a system that is better than a monolithic software system. Hence, tools resulting from novel research results can be quickly verified and put into common use.

Our approach is efficient because it describes digital evidence generally, rather than just sequences of bytes located within the source disk image. We do not propose to tag all data, only the specific evidence that is critical to a case: e.g., specific JPEGs, search terms, or files. In this paper, we argue for our approach by describing the DEX specification for three systems, and providing open-source implementations of DEX wrappers for each of the following:

- The Sleuth Kit's mmls and Darwin's fdisk partition table parsers.

- The Sleuth Kit's istat and our own kstat tool for parsing NTFS Master File Table (MFT) entries.
- EXIFtool and JHead for parsing Exchangeable Image File (EXIF), which is used to tag JPEGs with meta-information, such as camera model, date, and location.

Our code allows the chaining of the eight combinations of these tools and a comparison of the eight different outputs to quickly reproduce and compare one piece of EXIF evidence.

We begin with a review of related work.

2. Related work

Our paper is related to previous works on provenance, forensic formats that manage acquired data, and models of tools and investigations that produce acquired data. In this section, we review related work in those areas. We discuss related works in XML tree comparison in Section 3.4.

In sum, our work differs because of its focus on the evidence produced by tools, rather than the raw images that are acquired at the start of investigations. Note that in previous work, such raw, unprocessed images are often referred to as "evidence". However, in our context, we refer to evidence only as the specific items of interest that are harvested from an acquired image and presented in court. What we share with previous work is a concern with the reliability of tools, as this directly affects the suitability of evidence in court.

2.1. Provenance

All sciences rely on reproducible results, and therefore provenance has received attention from a variety of fields. Simmhan et al. (2005) offer an excellent survey of provenance techniques for the sciences, defined as any where the goal is a description of ancestral sources of a data product and the transformational processes and workflows that were used to derive the product from the source. For example, Foster et al. (Foster et al., 2002, 2003) propose a system called *Chimera* that records data provenance as a graph representing the derivation from the source dataset to a final data product. These graphs are defined and queried using a Virtual Data Language. Osterweil et al. (2008) similarly apply graphical notions of provenance to hydrological studies using derivation functions that include both tools and human operations. In digital forensics, provenance techniques can similarly be applied to report on transformations from an acquired raw image (i.e., a data source) to evidence (i.e., a data product); Process provenance metadata is not a standard part of investigations though it would address many concerns about the reproducibility of examinations. In comparison to these past works on provenance, DEX is a less general approach that is tuned to forensic investigations.

2.2. Image formats

A cryptographic hash of an entire disk image can be used to ensure that no bits have changed between acquisition and examinations. However, higher-level features are managed by advanced formats that have been proposed in recent work.

The popular EnCase format includes a header and CRC error detection codes on segments of the data. The Digital Evidence Bag (DEB) (Turner, 2005) format wraps raw images with metadata about an investigator, hash signatures of data, serial numbers of the original device, and a history of operations performed on the image. The Advanced File Format (AFF) (Cohen et al., 2009; Garfinkel et al., 2006) structures images into two layers. One layer contains segments that store distinct pages of the data acquired and metadata associated with each page. Metadata can include the serial number of the hardware, the time the image was acquired, and bad block lists. The second layer is the management structure for these segments and allows quick seeking and other performance-directed features. The description by Garfinkel et al. of AFF (Garfinkel et al., 2006) provides a good survey of many other image formats, including EnCase, FTK, ILook, and more.

These formats are focused on enhancing the performance of searching, carving, or otherwise analyzing an image. In contrast, we are focused instead on the process provenance of the evidence that is harvested. We assume that the raw image is maintained by one of these formats so that bytes associated with particular evidence can be extracted when needed. That is, our format does not store any data that is significant in size; instead a cryptographic digest is stored as a uniquely identifying record.

2.3. Tool abstractions

Related works on tool abstraction are focused more on the tools used to acquire evidence and less on the analysis of acquired data, which is our interest.

Carrier (Carrier, 2003) proposes a series of requirements for forensic tools: they must be usable, comprehensive, accurate, deterministic, and verifiable. These requirements are based on abstraction layers mirroring real systems, including physical media, media management, file systems, applications, network, and memory. Carrier proposes modeling each layer as handling a particular set of inputs, using a rule set, and producing a particular set of outputs. The layers are arranged hierarchically. With the goal of producing statistics on reliability for the courtroom, Carrier assigns two types of error rates to each layer. Tool implementation errors are flaws in a tools design or code that mismatch with a given specification, and they are detected via the process of *verification* (Aviževenis et al., 2004). Abstraction errors are introduced when a tool's representation of a system is not accurate. Carrier's goal was to offer a model of tools, their requirements, and their interactions. Our work builds on this foundation by proposing methods of canonically describing, verifying, and comparing evidence that is output by the tools used by investigators.

Gerber and Leeson (2002) propose developing a model significantly more detailed than Carrier's, based on a functional language that also describes the transitions of data between layers of abstraction in a computer system. The system begins with the physical medium and the electronics used, including low level CPU and memory functions. Their model would describe more than what is required to understand the data found on a system. It would be sufficient for the implementation of hardware or software that comprise a particular system. This level of detail might capture differences overlooked by coarser abstractions. For example, the Linux and Windows implementations of NTFS may treat data slightly differently, particularly because NTFS is not a published standard; and it is important for forensic investigators to note these differences. However, we maintain that the Gerber and Leeson approach is too heavyweight to be practical. It would be a challenge to capture every aspect of a live system in a manner as detailed as modeling every line of code and the electronics used. Our approach differs because we describe the output of forensic tools rather than the full details of the systems they acquire data from.

Pan and Batten (Pan and Batten, 2005) also builds on Carrier's abstraction layers to focus on reproducibility of acquiring an image. They note, for example, that reproducibility requires that data not be overwritten. In contrast, our focus is on the reproducibility, comparison, and verification of evidence harvested from such an image.

Finally, in an approach similar to ours, Alinka et al. (2006) propose basing an entire forensic investigation on a database of XML descriptions of tool outputs. Tools are wrapped to produce XML-tagged data that is stored in a single database. Their approach also allows for the easy serializing of tools. However, their goal is to improve the management of data parsed from an acquired image by storing it in a single database, as well as improving the efficiency of queries to the database by relying on XML. Independently of our work, Garfinkel (2009) recently proposed a system called *fiwalk* that produces a single XML structure from an acquired image that represents all file system and document artifacts contained within. Fiwalk is intended to be the basis of a range of other tools — hypothetically, one such tool could be designed to enable evidence provenance.

3. Design and implementation

In this section, we describe both our design and our open-source implementation of DEX, as well as our intended use of the system.

DEX is designed to meet two goals. First, a DEX description and the raw image file should be sufficient for reproducing evidence. Second, DEX descriptions should be sufficient for identifying differences between two separate investigations of the same raw evidence. We also expect that DEX can be used as a tool for improving tool reliability. It can be used to validate a tool or set of tools against known test cases, and similarly, it can be used as part of an NVP comparison. By tool reliability, we mean the tool consistently provides correct service against a specification (Aviževenis et al., 2004). Our focus is not the integrity of image acquisition or the examiner's chosen specification and analysis. We assume that images have been acquired properly, or at least that we cannot detect failures. We assume that tools are combined honestly and in a best effort to meet known methods of analysis, but we do not assume they are reliable. That is, if a tool outputs a DEX record, it does not imply it has so correctly. Finally, we assume that the specification's rules for parsing data are deterministic and not open for interpretation. While it is beyond our scope to provide a specification, we believe DEX can be helpful by making an examiner's use of a specification more explicit.

3.1. The details of DEX

A DEX file is an XML-based description of *objects* of interest — volumes, files, JPEGs, etc. — found in a raw image file. Each of the objects is composed of *records* that describe either: (i) data of interest, e.g., strings, binary data, timestamps, etc.; or (ii) data that manages other objects, e.g., an MFT entry is a record that contains the run list of a file, while the description of the file's contents is a separate object. Additionally, these objects record the provenance of the object. Here we use the terms object and record to refer to our abstraction of a system. Within a DEX file, an *element* refers to the actual XML data structure denoted by start and end tags of the same name. Elements may describe either objects or records.

In raw images, objects rarely appear sequentially and the correct ordering of fragmented bytes can be determined only by parsing (or carving) through a series of layers: partition tables, file systems, application headers, and so on. A DEX file represents the results of such parsing. All objects are listed in a flat series as the outer-most elements in the file, and records appear as elements inside their associated objects. Every DEX object has a pointer to the parent object that was parsed or carved in order to assemble the relevant bytes in the correct order. Records are implicitly created by parsing their enclosing objects. Each parent pointer is specified in the XPath language,[1] which is a standard format for addressing parts of an XML document.

An example of our approach is illustrated in Fig. 1. Each object points to the object that was parsed in order to locate it. In the figure, Volume 1 appears as a valid entry in the partition table, so it's location is described as entry 0. On the other hand, Volume 2 was carved from the raw image. Similarly, one EXIF record was found as part of a file referenced in the MFT, while another is part of an attachment to an email. A different example follows, showing a simplified DEX description of an MFT entry.

```
<DEXroot>
<DiskImage MD5 = 'abc123'>
  <Filename>image.dd</Filename>
</DiskImage>
<MasterFileTable MD5 = 'def456'>
  <ParentPtr>
  /DEXroot/DiskImage[@MD5 = 'abc123']
  </ParentPtr>
  <CommandLine>
  istat -o 63 image.dd 564
</CommandLine>
<entryAddress address = '564'>
    <DATA Resident = 'Non-Resident'>
    4992
    </DATA>
  </entryAddress>
</MasterFileTable>
</DEXroot>
```
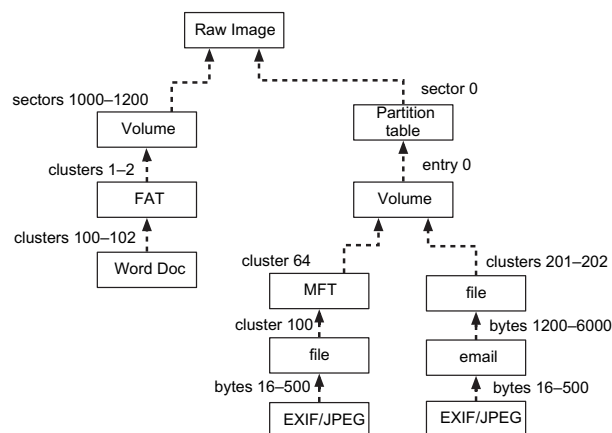

Fig. 1 – A diagramed version of a DEX description of evidence.

Each DEX starts with a raw image file object (<DiskImage>). For simplicity we show only the cryptographic digest, but other fields, such as acquisition date and tool, are possible. In our example, the description denotes only a single entry of interest and not the entire MFT. In this example, the MFT and the entry were parsed from the raw image directly, thus no partition table appears in the DEX. Instead, the DiskImage serves as the parent of the MFT.

Each object contains a cryptographic digest of the sequence of bytes specific to that object. For a DiskImage, the digest is calculated over the entire image. For the MFT, it is the bytes on the disk where the MFT itself is stored. A full DEX file appears in Appendix. We discuss how to compare two DEX files in Section 3.4.

We require that each object minimally include.

- A *cryptographic digest* (e.g., MD5 or SHA-1) of the bytes that were parsed to create the records contained with. The digest serves as an integrity check, and also to uniquely identify this object among all children of the parent. In the case that the object size is zero or trivial, we enforce a unique numbering among its siblings through comparison of record fields.
- *Pointer* elements link to other objects in the file that were parsed or carved to locate the object during the investigation. Pointers are specified as XML XPaths. The DEXRoot is the only object in the file with no ancestor.
- *Record* elements denote information available from parsing the object's data. Not all data that can be parsed need to be included. For example, a single entry from an MFT may be sufficient for an investigation. Fields are only essential to include if they are pertinent to an investigation or are critical for extracting descendant objects (such as run lists in an MFT entry); as we discuss later, the digests determine if two objects represent the same bytes in a disk image, regardless of how they are parsed, even if the parsing results differ.

Any record can optionally be replaced as a digest to reduce the DEX file size. For example, the portion of a JPEG encoding

---

[1] http://www.w3.org/TR/xpath.

an image is likely to be too large to reasonably store in the DEX, but the contents of specific EXIF tags can appear efficiently in the DEX itself.

• *Informational* elements contain ancillary data, such as the local pathname of an image, the tool used to create the object, version information, and other information about provenance or chain of custody if desired. Such data does not affect whether two DEX evidence files are considered equivalent descriptions of the same data.

An existing DEX file can be appended when another tool is used. For example, a partition table object can be added to our example above. In that case, a separate effort would point the MFT object to the partition table appropriately. However, we specifically do not allow two descriptions of the same object from different tools. For example, one file cannot contain both the mmls and fdisk parsing of the partition table. If both are desired, they would appear as separate files. Our rule implies that istat and kstat could not be used for separate entries inside one MFT object.

3.1.1. XML size and DEX scalability

DEX records objects and their provenance in an XML tree. Although XML is a verbose format, we expect that scalability will not be a major impediment to DEX deployment for the following reasons. First, various binary XML formats such as Fast Infoset[2] dramatically reduce both XML space requirements and parsing overhead. Moreover, whether binary- or text-based, XML compresses well when long-term storage is required. In either case, DEX markup of a tool's output will add a constant factor space overhead to existing tool output; we believe this overhead to be acceptable. Second, a DEX file need not contain all possible outputs of all tools, but only the output of interest — separate sub-trees that do not share cross-references need not be included in a DEX file if they are irrelevant to the task at hand. Finally, for investigations that manage massive amounts of evidence, we expect the DEX records to be managed by a larger system, such as Alinka et al. (2006).

3.1.2. Standardizing DEX

What determines our abstraction? When is a system abstracted to a record rather than an object? Enclosing an element inside another is equivalent to listing them separately and providing a pointer from one to the other. That is, there are many equivalent representations of the same evidence file. Our design of DEX has outer elements that correspond to the major abstraction layers and separate applications (Carrier, 2005) — partition tables, MFTs, files, EXIF data, etc. — but it is not the only valid scheme. Our choice of objects is related to the dynamics of comparing two DEX files, which we discuss in Section 3.4.

We intend to release a schema that defines DEX files, though we intend to delay this release until after our tools

---

[2] https://fi.dev.java.net.

have been public for some time. Writing out such a schema is not difficult, but it may calcify the DEX format prematurely. We plan to release a schema once we have feedback from the community on DEX and its design.

3.2. Three layers of wrapped tools

As a concrete prototype for our proposals, we implemented wrappers for six programs at three different system layers: mmls and fdisk for parsing partition tables; istat and kstat for parsing NTFS MFT entries; and EXIFtool and JHead for parsing Exchangeable Image File (EXIF) data in JPEG files. EXIFtool is available from http://www.sno.phy.queensu.ca/~phil/exiftool. JHead is available from http://www.sentex.net/~mwandel/jhead.

We picked these three layers because together they stand in contrast to how evidence is presented now. Typically evidence is currently presented as a written description of the EXIF data, along with a narrative describing how the evidence was acquired and which tools were used. A corresponding DEX file describes the EXIF data, but also describes the parsing and the tools behind the parsing that led to that EXIF data. Unlike the typical narrative format, the DEX provides a machine-readable record of forensic data acquisition, thus enabling the exchange, comparison, and validation of evidence and tools. The complete DEX file appears in Appendix and a diagram of the file appears in Fig. 2.

From the top layer of abstraction, an EXIF object contains records that are relevant to a case: a camera serial number and geographic information. The EXIF object also contains a pointer to the File object from which the EXIF data was parsed. The File object points to the MFT entry that described it within the image. The MFT entry describes the run list of clusters that can be concatenated to compose the JPEG File containing the EXIF data. The MFT entry object also contains MAC timestamp records and other metadata of forensic interest. The parent pointer of the MFT object is
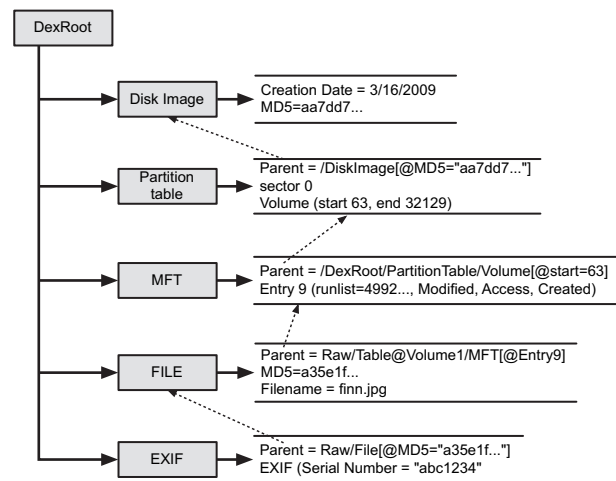

Fig. 2 – A diagram of the DEX file that appears in Appendix A. Solid lines represent enclosed elements. Dotted lines show the destination of Parent pointers.

the partition table object, which contains records listing the file system type (NTFS) and the sectors composing the file system volume. Finally, the partition table object points back to the raw disk image. At all levels, cryptographic digests are created and used to verify that data originated from the disk image.

Below, we describe each layer in more detail, and we use each layer as an opportunity to describe problems we encountered and solved in our design of DEX.

Our Java implementation of DEX is open source and freely available. The project can be downloaded from http://prisms.cs.umass.edu/forensics. The concepts we propose in this paper are not reserved intellectual property. Excluding comparison functions, all aspects of the implementation have a constant complexity and are straightforward to implement.

3.2.1. Partition tables

Partition tables are perhaps the most straightforward records that can be found on an acquired image of a hard drive. The partition table entry has only a few fields. However, the output of tools can vary even in this simple scenario: because fdisk is not a forensic tool, it details only volumes that are in the table. On the other hand, the output of mmls is closer to a map of the sectors on a disk — in particular, its output includes sectors that are not allocated in the partition table. Our wrapper includes the full information from each tool, respectively, in the two DEX files produced. Accordingly, when the two outputs are compared, output from one will not appear in the other.

From even this straightforward example, two potential problems arise. First, we found that outputs of different programs can be formatted differently. For example, some version of fdisk output information in hexadecimal notation. To address this potential inconsistency, we paid careful attention to storing data in a standard format.

Second, we found that we needed a more subtle comparison function than simple string equality across XML elements. In the case of partition tables, some meta-data is described inconsistently across tools — in particular, the partition type description (e.g., NTFS, NTFS/HPFS or Microsoft NFTS) is tool-specific, but it need not be identical for tools to have produced output we considered equivalent. Some comparisons are less straightforward, such as the extraction of particular fields within an MFT, or the potentially ambiguous results in reconstructing deleted files. Clearly, these sorts of comparisons depend upon the exact details of a given layer of abstraction. Thus, we require and implement layer-specific comparison functions, capable of differentiating at as fine a level as is reasonable between identical, equivalent, and different objects and entries within records. We believe this requirement is a reasonable one: It prioritizes practicality over total generality, and we expect that there will be a relatively small number of fields per layer that will need special comparison logic.

3.2.2. Master file table entries

In addition to The Sleuth Kit's istat, we used a custom MFT parser written by colleague Michael Krainin called kstat, which was implemented from Carrier's description of NTFS (Carrier, 2005). kstat is similar to istat but more limited in its output. For example, it does not parse files that have attributes that will not fit into one MFT entry. It does parse $STANDARD_INFO, $FILE_NAME, and $DATA attributes, and provides the run lists for attributes that are non-resident.

Parsing MFT entries illustrates why we use a pointer structure rather than nesting objects inside higher layer objects. There may be more than one NTFS partition on a disk, and pointers provide an unambiguous way to associate a MFT with a partition. On a related note, pointers provide also a flexibility for dealing with incomplete or missing data. For example, a disk image missing the partition table, such as an image of a partition only, will be identified as the parent of the MFT it contains. The notion of pointing to the most descriptive parent of the current object is useful in other contexts as well. For example, files that have been deleted may appear in the MFT and include an appropriate parent pointer, or they may not if the associated MFT entry has been overwritten, in which case the parent will point to the associated partition table entry or the disk image object, as appropriate.

3.2.3. File and EXIF data

Many metadata extraction tools such as EXIFtool are designed to read files stored on the local file system, not carved sequences of bytes for a disk image. Thus, we required the abstraction of a File object, which serves to link MFT entries and files created by extraction utilities such as The Sleuth Kit's icat. File entries in DEX are thus the union of actual files pulled from parsing file system on the image and carved files from any source, such as undelete utilities. These entries serve to keep the sources clear to both a machine and a human reader of the DEX.

Much like File objects, EXIF objects serve to organize and make clear the source of what is ultimately a sequence of bytes extracted from the file. By parsing and labeling as entries these bytes, we ascribe semantic meaning to them in a systematic, tool-independent fashion. Again, this requirement stems from practical considerations, as we expect that there is a limited set of file-metadata-level objects that are in common will need to be specified in a tool-independent fashion. In fact, when the parsed metadata is in a highly structured format, as is the case in EXIF, wrapping new tools is trivial.

3.3. DEX implementation

We have implemented tools for creating, extending, and comparing DEX files as a Java library and a set of wrappers over command line forensic tools. The core of our design is a library of classes that captures the abstraction of forensic objects, their attributes, and the relationships between them. Along with a small amount of utility code, these classes reduce the problem of wrapping forensic tools to parsing the tools' input and output, and instantiating Java objects corresponding to their layer of abstraction.

The library of classes also includes context-sensitive comparison functionality. As described in Section 3.4, different objects have different requirements for comparison. Thus, we require that when new objects are added to the abstraction, the comparison functions on the corresponding classes be updated. The wrapping of new tools at a given level of abstraction may also require these updates, particularly if a new tool exposes metadata not discovered by existing wrappers. Updates to the comparison function are straightforward, and we expect that authors of new tools and DEX wrappers will choose to perform such updates. Our current approach for unhandled records and entries is conservative: In the former case, the comparison function throws an exception, and in the latter, the entire record is flagged as unique to its DEX file.

3.4. Comparing investigations via DEX

Using DEX, two adversarial examiners can exchange and compare the results of investigations in a precise fashion that is independent of the tools used. Rather than agreeing upon the tools used, they need to only agree that the specification DEX adheres to is correct.

The comparison of XML trees is a well studied problem, often called tree-matching or change detection. Our design is easily able to leverage previous work to avoid complex versions of the problem. In particular, a comparison between two *unordered* XML trees has been shown to be NP-Complete in the most general case (Zhang et al., 1992).(An unordered tree is one where the left to right order of sibling elements does not matter.) Many heuristics for comparing unordered trees have been proposed that provide better running times. For example, using special characteristics of XML trees not considered in the general case, xdiff (Wang et al., 2003) has a $O(n \log n)$ complexity for documents of n elements.

Shorter running times are possible with ordered trees. Fortunately, we can impose an ordering on our output. Every object and record has a defined sorting key. For objects, we use the digests. For records, we sort field alphabetically, and we require the use of the byte offset if no other key is inherent to the data structure.

There are several existing algorithms for discovering differences between two ordered trees. The most general algorithms output the series of "patches" required to match two trees. For example, the treePatch XML comparison library produces a list of such patches for two XML trees. The tree-Patch algorithm is efficient at $O(le + e^2)$, for a tree of l leaf nodes and where e is the total number of patch operations on sub-trees (Komvoteas, 2003; Peters, 2005). treePatch represents results using a custom language describing inserts, deletes, updates, and moves.

Our current implementation is also linear in time but does not compute patches. Starting from the DEXroot, we compare all outer elements (i.e., objects) in each DiskImage present. Because each element is uniquely identified and only needs to be compared once, we do not need to perform a comparison of all pairs. Hence our algorithm is $O(n + m)$ for two trees with n and m objects, respectively. We lack descriptive output: we denote which elements differ, but not how the trees differ

according to the edit operations provided by treePatch. We output which records per object appear in one or both elements in the compared trees. In future versions, we plan to include the treePatch XML comparison library for more descriptive output.

One difficulty in DEX comparisons is that comparison functions for each type of element have to be customized, to some degree. Some elements may not appear identical but are equivalent. For example, as noted in Section 3.2.1, partition table parsing tools can output a descriptive string of a particular volume in a partition table. In fact, the only accurate description of the partition type is the numeric type identifier. Generally, then, fields in a DEX file must be represented in a canonical form and other fields that are purely informational must be identified and ignored by the comparison function, lest the DEX comparison falsely report differences when there are none. We note that such errors are always conservative, in that actual differences will not be overlooked, but differences in information fields may be mistakenly flagged. We do not aim to automatically fix such errors.

3.5. Testing and comparing output across tool chains

It is common practice today for an examiner to present results once as found by Guidance EnCase software and once as found by Access Data Forensic Tool Kit. Both have been accepted as valid tools by courts. The theory is that since the tools are developed independently, the evidence must be correct — that notion is the basis for N-version programming. Using DEX's standardized outputs, a combinatorial number of tool chains can be used to verify a specific piece of evidence. If all combinations of tools agree, then our confidence in the correctness of the implementations, and the evidence they document, increases. We assume that comparison of DEX files is feasible, as discussed in Section 3.4.

For testing in general, we require that a specification for a tool's functionality exist independent of any particular tool. By creating and executing test cases based upon this specification, we can increase our confidence in the correctness of an implementation. DEX files can be developed as part of standard test cases, instead of prose descriptions, such as those provided by the Digital Forensics Tool Testing Images site.[3] Furthermore, fabricated DEX files can represent cases where tools should detect errors or impossible scenarios if operating correctly. While DEX may make the description and use of test cases easier, a prominent limitation is that it does not make it easier to select and design test cases. As is the case in all testing, such tests can reject incorrect implementations, but cannot prove program correctness.

A form of NVP generalizes this procedure across multiple tools. For NVP to be valid, the tools must be developed independently from a known specification. For example, tools that were implemented by comparing output against The Sleuth Kit (TSK) during coding are not independent, as they might repeat the same implementation mistakes. On the other hand,

---

[3] http://dftt.sourceforge.net.

tools that are designed by reading Carrier's book (Carrier, 2005) can be compared against TSK tools as the book is a specification. Carrier's book or other specifications may have errors in validity, but such errors are not related to software reliability. Moreover, NVP cannot discover errors that exists in two tools despite independent development (Knight and Leveson, 1986).

In order to perform NVP, we require a method for extracting evidence from an image using the output from one tool in the chain as input for the next tool in the chain. We compose our code's ability to chain tools together and the ability to compare DEX outputs to implement a form of NVP. For each position i in the chain of tools, we use each tool that is appropriate to position i, while holding constant the other tools in the chain. For a given test case, each resulting DEX file should be identical.

We performed the 8-way comparison of the three-tool chain each with two alternatives. The results are as expected (i.e., equivalent), and are not shown for brevity. The appendix shows the output of the fdisk-istat-icat-JHead tool chain.

In our testing set up, these extraction tools are dd and icat. While dd performs simple extraction of a range of bytes, icat performs parsing, and therefore, we have wrapped the program with our DEX library. When called, it produces a DEX element describing a <File> object. The file object points to an MFT entry's run list and its digest record is based on the extracted data. The filename is a name based on the MFT entry; in the case that a similar tool carves a file, the filename can be a fabricated unique value. Note that because icat is parsing data, we could have an alternative in the tool chain, however, our test is not invalid without one.

3.5.1. Amalgams

We note also that using these same methods, and by using DEX as a common output format, a series of small tools can be combined together in a best-of-breed amalgam. These combinations potentially form a system that is better than a monolithic software system. Hence, tools resulting from novel research results can be quickly verified using NVP and put into common use.

4.   Conclusion

In this paper, we have described the importance of a standard, open format for digital evidence provenance, both for description and comparison of particular pieces of evidence as well as for tool interoperability and validation. We have presented DEX, an XML format for recording digital evidence provenance. We also have made public an open-source library that supports creation and comparison of DEX files and set of wrappers for several existing forensic tools.

Our future work will be governed by the community's response to our proposal, but we hope to add more features and functionality to DEX, with the goal of producing a practical and usable specification and software. In particular, we want to wrap more tools. For example, we see no impediments to creating objects and records that are appropriate for network traces, which follow a layered model as well. Further, as more tools and layers are wrapped by DEX, we plan to formalize the XML Schema that defines DEX files. We plan to develop tools to visualize the differences between DEX files, as well as a set of standard test cases to validate new tools for particular layers.

Appendix A.
An Example DEX File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DEX_root>

<DEXroot version="0.0">
  <CreationDate>2009-03-16 22:26:25</CreationDate>
  <DiskImage MD5Sum="6aa7dd7aa21061cca79e1d85cc1a8450">
    <Filename>disk-image</Filename>
  </DiskImage>
  <PartitionTable
ParentPtr="/DEXroot/DiskImage[@MD5Sum=6aa7dd7aa21061cca79e1d85cc1
a8450]">
    <SectorSize>512</SectorSize>
    <Offset>0</Offset>
    <Version>Darwin 9.6.0 Darwin Kernel Version 9.6.0: Mon Nov 24
17:37:00 PST 2008; root:xnu-1228.9.59~1/RELEASE_I386
i386</Version>
    <CommandLine>fdisk -d disk-image</CommandLine>
    <Volume>
      <StartSector>63</StartSector>
      <EndSector>32129</EndSector>
      <Description />
      <Type>7</Type>
    </Volume>
  </PartitionTable>
  <MasterFileTable
ParentPtr="/DEXroot/PartitionTable/Volume[StartSector=63]">
    <Version>The Sleuth Kit ver 3.0.1</Version>
    <CommandLine>istat -o 63 disk-image 27</CommandLine>
    <entryAddress address="27"
MD5sum="f7c3c7307accce6ef2797d7beb9ecd7e">
      <MFTEntryHeader>
        <Entry>27</Entry>
        <Sequence>1</Sequence>
        <LogFileSequenceNumber>0</LogFileSequenceNumber>
    <Links>1</Links>
      </MFTEntryHeader>

      <STANDARD_INFORMATIONAttribute>
        <Flags />
        <OwnerID>0</OwnerID>
        <Created>Wed Mar 11 09:38:28 2009</Created>
        <FileModified>Wed Mar 11 09:38:28 2009</FileModified>
        <MFTModified>Wed Mar 11 09:38:28 2009</MFTModified>
        <Accessed>Wed Mar 11 09:38:28 2009</Accessed>
      </STANDARD_INFORMATIONAttribute>
      <FILE_NAMEAttribute>
        <Flags />
        <Name>finn.jpg</Name>
        <ParentMFTEntry>5</ParentMFTEntry>
        <Sequence>5</Sequence>
        <AllocatedSize>0</AllocatedSize>
```

```
<ActualSize>0</ActualSize>
<Created>Wed Mar 11 09:38:28 2009</Created>
<FileModified>Wed Mar 11 09:38:28 2009</FileModified>
<MFTModified>Wed Mar 11 09:38:28 2009</MFTModified>
<Accessed>Wed Mar 11 09:38:28 2009</Accessed>
</FILE_NAMEAttribute>
<Att>
<STANDARD_INFORMATION Resident="Resident" />
<FILE_NAME Resident="Resident" />
<SECURITY_DESCRIPTOR Resident="Resident" />
<DATA Resident="Non-Resident">2517 2518 2519 2520 2521
2522 2523 2524 2525 2526 2527 2528 2529 2530 2531 2532 2533 2534
2535 2536 2537 2538 2539 2540 2541 2542 2543 2544 2545 2546 2547
2548 2549 2550 2551 2552 2553 2554 2555 2556 2557 2558 2559 2560
2561 2562 2563 2564 2565 2566 2567 2568 2569 2570 2571 2572 2573
2574 2575 2576 2577 2578 2579 2580 2581 2582 2583 2584 2585 2586
2587 2588 2589 2590 2591 2592 2593 2594 2595 2596 2597 2598 2599
2600 2601 2602 2603 2604 2605 2606 2607 2608 2609 2610 2611 2612
2613 2614 2615 2616 2617 2618 2619 2620 2621 2622 2623 2624 2625
2626 2627 2628 2629 2630 2631 2632 2633 2634 2635 2636 2637 2638
2639 2640 2641 2642 2643 2644 2645 2646 2647 2648 2649 2650 2651
2652 2653 2654 2655 2656 2657 2658 2659 2660 2661 2662 2663 2664
2665 2666 2667 2668 2669 2670 2671 2672 2673 2674 2675 2676 2677
2678 2679 2680 2681 2682 2683 2684 2685 2686 2687 2688 2689 2690
2691 2692 2693 2694 2695</DATA>


</Att>
</entryAddress>
</MasterFileTable>
<File
ParentPtr="/DEXroot/MasterFileTable/entryAddress[@address=27]"
MD5Sum="7a35e1fb89cd05b8465d97f6b402404c">
<Version>The Sleuth Kit ver 3.0.1</Version>
<CommandLine>icat -o 63 disk-image 27</CommandLine>
<Filename>finn.jpg</Filename>
</File>
<Exif
ParentPtr="/DEXroot/File[@MD5Sum=7a35e1fb89cd05b8465d97f6b402404c
]">
<Version>Jhead version: 2.87   Compiled: Mar 11
2009</Version>
<CommandLine>jhead finn.jpg</CommandLine>
<CameraMake>Canon</CameraMake>
<CameraModel>Canon PowerShot A720 IS</CameraModel>
<DateTime>2008:10:19 09:38:59</DateTime>
</Exif>
</DEXroot>
```

REFERENCES

Alinka W, Bhoedjanga R, Bonczb P, de Vriesb A. XIRAF –XML-based indexing and querying for digital forensics. In: Proc. DFRWS; 2006.

Avižeienis A. The N-version approach to fault-tolerant software. IEEE Trans Software Eng 1985;SE-11(12):1491–501.

Avižeienis A, Laprie J-C, Randell B, Landwehr C. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Dependable Secure Computing 2004;1(1):11–33.

Battistoni M. Netherlands forensic institute develops and publishes open source software. Technical Report. Open Source Observatory and Repository, http://hdl.handle.net/2038/1579; Dec 2004.

Carrier B. Open source digital forensics tools: the legal argument. Technical report, @stake Research Report, Oct 2002.

Carrier B. Defining digital forensic examination and analysis tools using abstraction layers. Int J Digital Evidence Winter 2003;1(4).

Carrier B. File system forensic analysis. Addison-Wesley; 2005.

Cohen M., Garfinkel S, Schatz B. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, Arbitrary Information and Forensic Workflow. In: Proc. Annual DFRWS Conference; August 2009.

Foster I, Vöckler J, Wilde M, and Zhao Y. Chimera: A virtual data system for representing, querying, and automating data derivation. In: Proc. Int. Conf. on scientific and statistical database management, July 2002.

Foster I, Vöckler J, Wilde M, Zhao Y. The virtual data grid: a new model and architecture for data-intensive collaboration. In: Proc. Conf. on Innovative Data Systems Research (CIDR); Jan 2003.

Garfinkel S. Automating disk forensic processing with SleuthKit, XML and Python. In: IEEE workshop on systematic approaches to digital forensic engineering, May 2009.

Garfinkel S, Malan D, Dubec K, Stevens C, Pham C. Disk imaging with the advanced forensics format, library and tools. In: Proc. IFIP Intl Conf on Digital Forensics; Jan 2006.

Gerber M, Leeson J. Shrinking the ocean: formalizing I/O methods in modern operating systems. Int J Digital Evidence 2002;1(2).

Kenneally E. Gatekeeping out of the box: open source software as a mechanism to assess reliability for digital evidence. Va J Law Tech 2001;6(13).

Knight J, Leveson N. An experimental evaluation of the assumption of independence in multi-version programming. IEEE Trans Software Eng 1986;12(1):96–109.

Komvoteas K. XML Diff and patch tool. Master's thesis, Heriot-Watt University, http://treepatch.sourceforge.net; Sept 2003.

Osterweil L, Clarke A, Ellison, Podorozhny R, Wise A, Boose E, Hadley J. Experience in using a process language to define scientific workflow and generate dataset provenance. In: Proc. Intl Symp. on Foundations of Software Engineering, 2008. p. 319–29.

Pan L, Batten L. Reproducibility of digital evidence in forensic investigations. In: Proc. DFRWS, 2005.

Peters L. Change detection in XML trees: a survey. In: third Twente Student Conference on IT; June 2005.

Simmhan Y, Plale B, Gannon D. A survey of data provenance in E-Science. ACM SIGMOD Record 2005;34(3):31–6.

Turner P. Unification of evidence from disparate sources (digital evidence bags). In: Proc. DFRWS, 2005.

Wang Y, DeWitt D, Cai J.-Y.. X-Diff: an effective change detection algorithm for XML documents. Intl Conf. on Data Engineering; 2003.

Zhang K, Statman R, Shasha D. On the editing distance between unordered labeled trees. Inform Process Lett 1992;(42)::133–9.

Brian Neil Levine is an Associate Professor in the Dept. of Computer Science at University of Massachusetts Amherst. He received a PhD in Computer Engineering from the University of California, Santa Cruz in 1999. His research focuses on mobile networks, forensics and privacy, and the Internet, and he has authored more than 60 papers on these topics.

Marc Liberatore joined the Dept. of Computer Science at the University of Massachusetts Amherst as a Research Scientist in January 2009. Previously, he served as a Mellon Postdoctoral Fellow at Wesleyan University in Middletown, Connecticut. He earned his Master's and PhD is Computer Science from UMass Amherst in 2004 and 2008, respectively. His research focuses on anonymity systems, digital forensics, peer-to-peer architectures, and disruption tolerant networking.