Refining Evidence Containers for Provenance and Accurate Data Representation

Bradley Schatz, Michael Cohen

Chapter 16

REFINING EVIDENCE CONTAINERS FOR PROVENANCE AND ACCURATE DATA REPRESENTATION

Bradley Schatz and Michael Cohen

Abstract It is well acknowledged that there is a pressing need for a general solution to the problem of storing digital evidence, both in terms of copied bitstream images and general information that describes the images and context surrounding a case. In a prior paper, we introduced the AFF4 evidence container format, focusing on the description of an efficient, layered bitstream storage architecture, a general approach to representing arbitrary information, and a compositional approach to managing and sharing evidence. This paper describes refinements to the representation schemes embodied in AFF4 that address the accurate representation of discontiguous data and the description of the provenance of data and information.

Keywords: Evidence containers, representation, provenance, tool interoperability

1. Introduction

One of the principal challenges in digital forensics is to deal with the rapidly growing volume and complexity of information that is the subject of investigations [4]. The acquisition and analysis of digital evidence are hampered by the lack of interoperability between forensic analysis tools; important forensic information is often unused by analysis tools because it is locked within proprietary file formats or free text. Access to case data is hampered by closed abstraction layers and the inefficiencies imposed by the need to manually copy data in order to process it with task-specific tools. Finally, the management of evidence is slowed by format conversion and storage bandwidth limitations.

Digital forensic practitioners have largely settled on the raw (`dd`) and Expert Witness Format (EWF) evidence storage formats for hard drive

images. This has enhanced the ability – in the storage forensics field at least – to acquire and analyze evidence using a variety of tools of commercial and open source lineage.

Such formats are, however, poor surrogates for the original evidence. A raw image fails to distinguish between sectors containing bytes with the value zero and those where a read error has occurred. A raw image does not record provenance-related information pertaining to the drive such as the drive geometry and drive configuration overlays, nor does it record the activities performed on the drive or image. The EWF format popularized by the EnCase tool is similarly limited; however, it does overcome some of the weaknesses of a raw image by recording limited meta information related to the image within the container itself.

In practice, provenance information describing the evidence and its outside context rarely becomes grist for the automated forensic mill, mainly because it is collected and recorded manually by investigators in a variety of formats, including handwritten free text and *ad hoc* file formats. New approaches are required to represent digital evidence, both in terms of raw bits and bytes on storage media (data) and information describing related artifacts, entities and analytic results.

In an earlier paper [9], we introduced the AFF4 evidence container format, which is designed to store arbitrary evidence images, context-related information and analysis results within a unified container format. This paper describes refinements to AFF4 that address the accurate representation of discontiguous data and describe the provenance of data and information.

2. Ideal Evidence Container

The ideal evidence container would present to the investigator a perfect surrogate of the original physical evidence, whether it is a hard drive, mobile phone flash memory or computer RAM. Such a container would fully describe the characteristics, behavior, content and context of the original evidence that it represents. These include:

- **Data Content:** Multiple streams (HPA, DCO); hierarchical data relationships (logical imaging); addressing windows (RAM holes, bad sectors); addressing schemes (block size, CHS/LBA); SMART status.
- **Physical Characteristics:** Make; model; serial number; interface (SATA, SCSI, etc.).
- **Context:** Environment in which the hard drive existed; case-related information.

- **Behavior:** Error codes related to bad sectors.

A perfect fidelity surrogate – even if it were technically feasible – would not be entirely desirable. Such a "virtual hard disk" would quickly frustrate the investigator by reliably imposing characteristics such as read retries on bad sectors, I/O bandwidth limitations and seek latency. Accordingly, the ideal container would sacrifice fidelity to satisfy orthogonal operational concerns such as:

- **Efficiency:** Storage space minimization; random access performance; I/O speed.
- **Authentication:** Cryptographic signing; hash storage.
- **Privacy:** Encryption; redaction.
- **Resilience:** Tolerance to underlying storage media failures.

3. Current State of Evidence Containers

From the original raw evidence format, evidence containers evolved to incorporate seekable compression [9], embedded authentication and integrity mechanisms such as hashes and CRC, and storage of a small number of fields for describing images. More recently, the demand for logical imaging has resulted in the emergence of a proprietary evidence container format that supports the storage of multiple streams of data along with file-oriented metadata (EnCase Logical Evidence File).

This current breed of commercial evidence containers does not address the characteristics of evidence sources. For example, hard drives may contain multiple address spaces, depending on whether features such as host protected areas or drive configuration overlay are enabled. Images of computer RAM require the consideration of holes in which no data exists. Furthermore, the evidence containers fail to address the storage of general information that is of relevance.

The research community has proposed a number of container formats. The Advanced Forensics Format (AFF) [10, 11] introduced the storage of arbitrary metadata within an evidence container, privacy via encryption and redaction, and resilience via fault tolerance. Digital evidence bags [15] store arbitrary textual information along with images in the same container. Sealed digital evidence bags [14] employ a composition framework for evidence containers based on a linked information model.

We recently introduced the AFF4 evidence container format [9], which defines an efficient, seekable, compressed storage format for multiple data object images, a novel and powerful data model that enables the composition of data objects from other data objects, and an information

Table 1. Comparison of representational capabilities of container formats.

	Data Representation							
	raw	**EWF**	**sgzip**	**LE1**	**DEB**	**SDEB**	**AFF1**	**AFF4**
Single Image Storage	Y	Y	Y	N	Y	Y	Y	Y
Multiple Image Storage	N	N	N	N	Y	Y	N	Y
Hierarchical Image Storage (Logical Imaging)	N	N	N	Y	Y	Y	N	Y
Addressing Windows (Discontiguous Images)	N	N	N	N	?	N	N	Y
Data Composition	N	N	N	N	N	N	N	Y
Seekable Compression	N	Y	Y	?	N	N	Y	Y
	Information Representation							
	raw	**EWF**	**sgzip**	**LE1**	**DEB**	**SDEB**	**AFF1**	**AFF4**
Metadata Storage	N	Y	N	Y	Y	Y	Y	Y
Arbitrary Metadata Storage	N	N	N	?	Y	Y	Y	Y
Arbitrary Information Storage	N	N	N	N	Y	Y	N	Y
Formal Information Model	N	N	N	N	N	Y	N	Y
Composable Information Model	N	N	N	N	N	Y	N	Y

representation approach based on a linked information model. Table 1 compares the characteristics of AFF4 and those of other evidence container formats.

4. AFF4 Data and Information Models

The general design goals of AFF4 are to provide an open and extensible evidence container that facilitates the storage, composition and sharing of arbitrary types of digital evidence, information and analysis results. AFF4 defines two interrelated models, one for representing and documenting information, and the other for storing, referring to and transforming bitstream data. The two models are linked by a naming scheme in which items of relevance are identified using globally-unique identifiers.

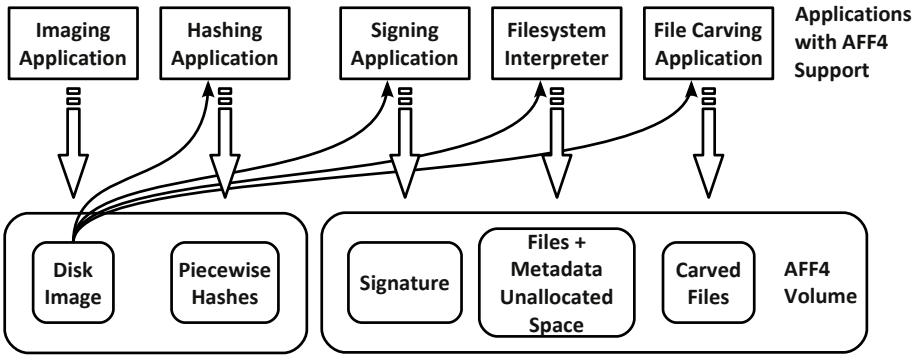

Figure 1. Layered application of forensic tools with the AFF4 container.

The AFF4 data model is specifically designed to facilitate the storing, sharing and referencing of data without imposing the storage bandwidth burden of copying data, while providing efficiency in terms of storage space, random access latency and I/O bandwidth. Likewise, the AFF4 information model is designed to facilitate the description of forensically-relevant information, including evidence, case context and general analytic results.

The "by reference" approach of the AFF4 architecture facilitates the successive layered application of discrete task-specific tools on the analytic results and data abstractions of tools operating at lower layers of abstraction. In the architecture, the analysis results of a tool are persistent when using the AFF4 information and data models in a new AFF4 container or in an existing container. This significantly differs from other approaches that either keep analysis results as intermediary structures in the working memory of a tool (as in monolithic applications like EnCase) or dump the results to an *ad hoc* (and likely non-machine-readable) document.

Figure 1 presents the flow of information and data between AFF4-aware forensic applications and AFF4 evidence containers. In this example, an examiner employs an AFF4 imaging application to create two disk images in a single container. Each image has two virtual address spaces overlaying it: one for the regular portion of the disk and the other that includes the HPA portion of the disk. The examiner uses the GUI of the tool to enter case-relevant and context-related information associated with the images; meanwhile, the imaging tool records provenance-related information obtained directly from the drives. Finally, the tool records the MD5 hash of the entire disk and the information related to media failures.

Upon returning to the laboratory, the examiner proceeds to preserve the evidence. A piecewise hash of the disk is created using SHA-256 or some other hash algorithm (this is performed in the laboratory because a hash algorithm such as SHA-256 is too slow to use in the field). The computed hash values are stored in a new container, which refers to the original container.

The investigator then uses a public key to cryptographically sign the information associated with the image, including the piecewise hashes. This signature is stored in the new container.

Automated processing of the image may involve a filesystem interpreter to create virtual file instances for each file in the image, instances for deleted files and instances for unallocated space. These are also placed in a new evidence container for subsequent consumption by other tools. The unallocated space is then processed by a file carver, the results of which are stored as virtual file instances in a new evidence container, resulting in zero copy file carving [12].

This scenario demonstrates the feasibility of the AFF4 architecture with respect to forensic imaging, piecewise hashing, signing and using filesystem interpreter applications.

5. AFF4 Naming Scheme

The fundamental premise of representing both data and information within AFF4 is that an object instance is created for any virtual or real-world entity – be it a disk partition or a suspect. The object instance acts as a surrogate for the real or virtual entity. Surrogates are identified by associating a globally-unique identifier with each entity. We use both uniform resource locators (URLs) and uniform resource names (URNs) as entity identifiers.

We define structured URNs, which we call AFF4 URNs, for identifying surrogates. An AFF4 URN is made globally unique by including a GUID in its form:

```
urn:aff4:195bdf58-1bc9-4ba4-9a9c-f1c312673fbf
```

We use a variant of the ZIP file format for a default container. Thus, an investigator who visually examines an AFF4 volume containing a disk image would readily identify AFF4 URNs in the following locations:

- The information segment, which is a file in an AFF4 container, contains a serialization of all the objects within the volume. The base attribute of the RDF serialization refers to the volume URN.
- Image file segments are present in a folder within the ZIP file, where the folder name is a filesystem-friendly encoding of the URN

that uniquely identifies the image. For example, a folder named `urn%3Aaff4%3A195bdf58-1bc9-4ba4-9a9c-f1c312673fbf` might refer to the image stream `urn:aff4:195bdf58-1bc9-4ba4-9a9c-f1c312673fbf/`. Archive filenames can be shortened relative to the volume URI. For example, the folder `diskimage` corresponds to the fully-qualified URI `urn:aff4:volume_URI/diskimage`.

- Information segments (described in the next section) contain URNs within their text.

URLs, which are typically used to uniquely identify a piece of terminology, may be found as text within information segments and maps. For example, we use the following URL to represent the concept of an "Image:"

```
http://afflib.org/2009/aff4#Image
```

6. Refining the AFF4 Information Model

The original AFF4 information model was inspired by the Resource Description Framework (RDF) [17], the data model that underlies the Rich Site Summary (RSS) feeds used by blogging software. The RDF data model facilitates object-oriented modeling with the key difference that objects in the RDF universe have unique names and properties, and the attributes and relationships of individual objects may be published in different documents.

While the original information model is intuitive and simple to implement, it suffers from several shortcomings:

- The method of mapping the information model to container segments (i.e., serialization scheme) results in inefficient storage when large numbers of information instances are described.
- The information model is not expressive enough to describe provenance-related information.
- The AFF4 serialization scheme is verbose, which makes it difficult to read.
- The lack of value types leads to ambiguity in interpreting values. Also, there is no syntactic means for distinguishing between values and URI references.

For these reasons and for standardization and interoperability, we abandon the *ad hoc* RDF variant and instead adopt RDF in its entirety as the information model. Under the new scheme, information is stored in information segments whose suffixes represent the RDF encodings.

For example, the name `information.turtle` in the AFF4 container, refers to an RDF serialization using the Turtle encoding scheme [2]. For conciseness and readability, we use Turtle as the default RDF serialization syntax. The URN of the serialization component is significant, with the path component being interpreted as the graph name in which the encoded RDF exists.

7. Provenance of Information

This section describes a general approach for expressing the provenance of statements in the AFF4 universe. Provenance statements are required to express information such as which tool generated which image or analysis products, and to sign statements.

The original AFF4 provides for provenance-related statements in the particular case of signing. However, the identity object, which implements signatures, presents some difficulties in the case of provenance statements:

- The semantics of the statement file in relation to its enclosing instance is inconsistent with the information model.
- Statements within a statement file are also made elsewhere within the container.
- Verifying the signatures of statements requires that the statements be in the exact order and syntax in which they exist in the signature file.

A general solution to provenance requires a method for referring to a set of RDF statements as a whole. Such statements about statements are called "reified statements" in the knowledge representation literature. A simple example is: Dick said "The serial number of the hard drive is `ZX322o91` and its hash value is `13343af423d`."

The subject Dick is making a statement covering two separate statements: "The serial number of the hard drive is `zx322o91`" and "The hash value of the hard drive is `13343af423d`."

A widely acknowledged problem of RDF is its limited ability to express reified statements [7]. Named graphs constitute a solution to the problem of reification in RDF, with the TriG language emerging as a syntax for encoding reified information [6]. A named graph is simply a collection of RDF statements that can be identified by an unambiguous name. Using the TriG syntax, the reified statement above can be expressed as:

```
1    @prefix G1:  <urn:aff4:19857a87-a190b2f87>
2    @prefix Hdd1: <urn:aff4:652e4027-27fab2941>
```

```
3    @prefix aff4: <http://afflib.org/2009/aff4#>
4    @prefix Dick: <urn:aff4:652e4027-27fab2941>
5
6    G1: {
7      Hdd1: aff4:serialNumber "zx322o91"
8      Hdd1: aff4:hash "13343af423d"
9    }
10
11   Dick: aff4:said <G1:>
```

In the listing above, Lines 1-4 define namespace identifiers that are substituted when they occur elsewhere in the document. For example, in Line 7, `aff4:serialNumber` is interpreted to mean the URL `http://afflib.org/2009/aff4#serialNumber`. This uniquely-identified vocabulary term is defined to have the meaning of a serial number. In Lines 6-9, `G1:` is the unique identifier for the named graph that contains the two statements referred to in our example. Named graph identifiers may be referred to in the subject and object parts of RDF statements. Finally, in Line 11, there is a single statement that refers to the named graph `G1:`.

Following this approach, we refine the semantics of the AFF4 information model to imply that for any information segment, the statements implied by interpreting the content of the segment are defined to exist within a named graph based on the following conventions:

- The graph name is the URN of the volume when the information segment is in the root of the volume.
- The graph name is the URN interpretation of the path when the information segment exists in a sub-path of the volume.

Consider, for example, an AFF4 ZIP container containing a ZIP file comment `urn:aff4:6cd61-52398e-4942ea` and the two information segments in the listing:

```
1    /information.turtle
2    /urn%3Aaff4%3A19d6cd61-598e-49ff/information.turtle
```

The RDF statements contained in the first segment would be interpreted to exist in a named graph with the URN of the AFF4 volume `urn:aff4:6cd61-52398e-4942ea`. The RDF triples contained in the second segment would be interpreted as being contained in the graph named `urn:aff4:19d6cd61-598e-49ff` after decoding the filesystem-friendly encoding.

Provenance-related statements employ the named graph semantics in their statements. Consider, for example, the recording of the provenance

of the information describing an image generated by the command line tool `aff4imager`. An abridged set of statements is presented in the listing:

```
1    @prefix G1: <urn:aff4:19857a87-a190b2f87>
2    @prefix G2: <urn:aff4:0a1fc78a-927bfacef>
3    @prefix T1: <urn:aff4:652e4027-ffff01199>
4    @prefix I1: <urn:aff4:9003027a-11199ffff>
5    @prefix aff4: <http://afflib.org/2009/aff4/#>
6
7    G2: {
8      T1: aff4:name "aff4imager"
9      T1: aff4:vendor <http://aff.org/>
10     T1: aff4:asserts G1:>
11     T1: aff4:type aff4:AcquisitionTool.
12     T1: aff4:version "0.2"
13     I1: aff4:type aff4:Image
14     I1: aff4:hash "3897450fa18094b13"^^aff4:md5
15   }
```

In this example, we define an instance `T1:` that represents the tool and an instance `I1:` that represents the image. The `aff4:asserts` predicate is used to specify that the tool "asserted" the information contained in the graph `G1:`.

By identifying instances of type `aff4:Tool` and then identifying the graph in which the statements are located, downstream consumers of AFF4 containers would be able to identify the tool that generated specific information and data. While it is not related to provenance, note that the type `^^aff4:md5` in Line 14 indicates the data type of the text preceding it within quotes. In this case, it indicates that the value of the `aff4:hash` predicate is the hex-encoded MD5 message digest of the image.

8. Authentication and Non-Repudiation

With a means for referring to sets of statements in place, the approach to authentication and non-repudiation of evidence can be been refined. We conceptualize the relationship of signing containers in a manner similar to the approach proposed in [6]. An identity remains a person or entity as in the earlier AFF4 implementation. A warrant graph is a set of statements that record the intentions or beliefs of an identity about another set of statements, whether it be asserting, denying or quoting. The identity vouches for the truth of the warrant graph by signing the graph with a public key.

The following listing contains a warrant graph that refers to the `aff4imager` information presented in the listing above:

```
1    @base: <aff4://1b056380-a0911-f06721>
2    @prefix G2: <urn:aff4:0a1fc78a-927bfacef>
3    @prefix G3: <aff4://19857a87-a190b-2f87ab>
4    @prefix A1: <aff4://502ffb11-00f10-7fcbaf>
5
6    G3: {
7      G2: aff4:assertedBy G3:
8      G3: aff4:hash "TljN2NiNzExMmEwM2MxNG"
                                ^^aff4:canonical-sha256
9      G3: aff4:authority A1:
10     A1: aff4:certificate A1:/cert.pem
11     G3: aff4:signature "XSAFfbgEL5C8vA1W/W-="
                                ^^aff4:canonical-sha256-rsa
12   }
```

The following observations can be made with regard to the listing:

- Line 7 refers to the graph `G2:` from the previous listing. This statement indicates that the warrant graph asserts the truth of `G2:`. Any number of named graphs may be asserted (or denied or other) within a warrant graph.
- The graph digest of graph `G3:` is stated in Line 8. For serialization independence, a graph digest is a message digest with the canonical form of a set of triples in a named graph rather than the serialized syntax. This facilitates the verification of the authenticity of the target graph. The graph canonicalization method is specified by the type parameter `aff4:canonical-sha256`, and is a variant of the graph canonicalization algorithm in [5] and the digest and signature methods defined in the XML signature standard [1]. The type parameter additionally indicates that the graph digest uses the SHA-256 digest on the canonical graph.
- Line 9 states that the identity `A1:` authorizes the warrant graph.
- Line 10 states that the public key certificate of `A1:` is found at the URN `A1:/cert.pem`.
- Line 11 states the signature of `G3:` (warrant graph). The type `^^aff4:canonical-sha256-rsa` indicates the method by which the signature was constructed, which was to take the warrant graph, canonicalize as above, take the SHA-256 hash, sign it using the RSA private key of the authority `A1:`, and then encode it using Base64.

The verification of a signed AFF4 container involves identifying a signed warrant graph, removing the `aff4:signature` statement from

the graph, canonicalizing the resulting graph, and then re-verifying the calculated SHA-256 RSA signature. Recalculating the graph digests of graphs asserted by the warrant graph further authenticates the information contained in the graphs.

The use of named graphs, graph digests and graph signatures facilitates the piecewise generation of authenticable and non-repudiable information in the AFF4 universe.

9. AFF4 Data Model

AFF4 defines two abstractions for storing and representing bitstream data: the Stream and the Map. Instances of each of these abstractions are identified by an AFF4 URI.

The lowest layer of abstraction for data storage in the AFF4 data model is the Stream, an abstraction of a contiguous, randomly-accessible byte sequence. AFF4 defines a number of implementations for storing and accessing Streams ranging from an efficient randomly-accessible compressed container to a flat raw file.

The Map abstraction similarly represents a contiguous, randomly-accessible byte array; however, it is composed of byte arrays from multiple stream sources. Defining virtual data objects as comprising portions of existing concrete data sources enables references to data objects within images (e.g., files) or data objects composed of multiple images (e.g., reconstructed RAID volumes). Maps are used as the fundamental building block for representing the data portions of files, partitions, unallocated space and reconstructions of virtual RAID volumes from images. Backward compatibility with EWF images is provided by creating a virtual image that maps to each compressed EWF segment.

10. Refining the AFF4 Data Model

AFF4 applications have revealed two shortcomings of the data model: (i) referring to subranges of data within a Stream is heavyweight, requiring the definition of a Map; and (ii) the data model does not support discontinuities that occur in evidence sources such as RAM and faulty hard drives.

10.1 Referring to Byte Ranges

AFF4 requires a means to refer to arbitrary address ranges (slices) within AFF4 data objects. This is useful for a number of applications, including the annotation of content in TCP streams with the time of transmission, documenting provenance-related features of the derivation

of an analytic product (e.g., file metadata), and describing the piecewise hash value of a chunk of an image.

An AFF4 slice provides the means to refer to a subrange within a URI. It is expressed by specifying the range within the fragment component of the URI:

```
URI#[offset:length]
```

The URI is a regular AFF4 Stream URI. The offset is a number that indicates the byte offset within the stream address range and the length is the number of bytes from the offset. For example, the slice URI:

```
urn:aff4:195bdf58-1bc9-4ba4-9a9c-f1c312673fbf#[512,128]
```

represents the address range from offset 512 to 640 in the URI.

```
urn:aff4:195bdf58-1bc9-4ba4-9a9c-f1c312673fbf
```

Note that the slice URI corresponds to an address range. In cases where the address range is backed by a stream of actual data, it also serves as a surrogate for the data within the bounds of the range. In the case where there is no corresponding data, it is a surrogate for the absence of data (i.e., an address space hole).

An example of a slice URI is demonstrated using a prototype piecewise hashing tool. Consider the following slice URI:

```
urn:aff4:f37648c1#[0,2048] aff4:hash "c35f2ba345"^^aff4:sha256
```

The interpretation of the slice URI (which has been truncated for presentation) is that the content of the byte range from 0 to 2048 of the stream `urn:aff4:f37648c1` has a SHA-256 value of `c35f2ba345`.

10.2 Representing Discontiguous Data

With the exception of the original AFF, existing forensic formats do not provide support for accurately imaging discontiguous data sources. Examples of discontiguous data sources include disks with read errors in certain sectors and the physical and virtual memory of computers with address space holes. For reasons of accuracy and completeness, it is important that the evidence container identifies areas of the Stream address space where there is no corresponding data, and potentially, the reason for the absence of data.

AFF4 provides a general solution to this problem by refining the semantics of the Map abstraction. Whereas the Map abstraction initially required that the target be a URI that resolves to a Stream or a Map, the refinement additionally allows the inclusion of a specially-defined URI as a target. Such a URI may indicate the characteristics of a byte range.

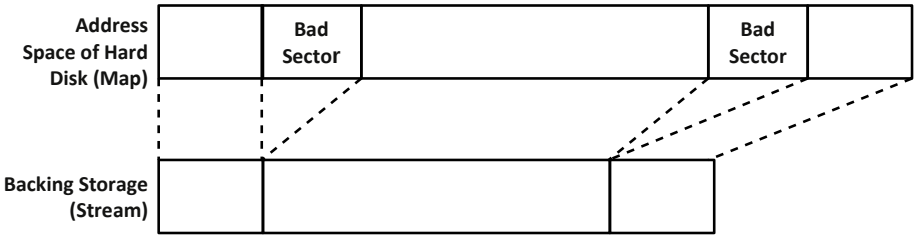

Figure 2. Address space mapping of a discontiguous evidence source.

For example, consider the imaging of a hard disk with a bad sector. We define a Map to represent the address space of the original evidence device and record the data content in a regular Stream. The Map records correspondences between the data content in the original device and the Stream. Figure 2 illustrates the mapping between the address space of an acquired hard drive with read errors and the underlying storage Stream. Note that the backing store stores valid data back to back, while the Map provides a view of the data with missing data represented as holes.

The address discontinuity that corresponds to the read error is given the target URI `aff4:UnknownData`. The following listing shows a Map segment that describes a discontinuity in this manner:

```
1    0,0,urn:aff4:da0d1948-846f-491d-8183-34ae691e8293
2    4096,0,http://libaff.org/2009/aff4#UnknownData
3    8192,4096,urn:aff4:da0d1948-846f-491d-8183-34ae691e8293
```

11. Representing Data Patterns

Storage media commonly comes from the factory with the data content of every byte set to zero. With hard drives rapidly increasing in capacity, it is often the case that large runs of data within images contain zeros ("zero data runs"). The AFF4 format reduces the storage impact of zero data runs by compressing the runs. However, due to the uniform chunking method, there is the potential for considerable repetition in sources containing large numbers of zero data runs.

Known data runs may be expressed in AFF4 using the slice URI and Map facilities. For zero runs, we define a special purpose URI with semantics similar to the UNIX `/dev/zero` to represent a Stream containing an unbounded number of zeros. This URI is `http://afflib.org/2009/aff4#ZeroFilledByteRange`.

Of relevance to referencing common data patterns is recent work related to Teleporter [16], which explores the efficient transmission of forensic images by sending certain data runs of the image with reference

to a standard corpus of files. Thus, in the case of a common file such as `ntfs.dll`, Teleporter would transport a fingerprint of the file rather than the entire data content. The receiver would then reconstitute the data content of the file from local sources.

12. Conclusions

The refinements made to the information and data models of the AFF4 evidence container format address the accurate representation of discontiguous data, help describe the provenance of stored evidence, and support authentication and non-repudiation of data and information by cryptographic signing.

Prototype implementations of AFF4 have been written in C, Python and Java. Our future research will integrate AFF4 with the PyFlag network forensic environment, the Volatility volatile memory analysis framework and the Sleuth Kit filesystem analysis tool. Also, it will ensure that AFF4 provides backward compatibility with the raw, EWF and AFF1 evidence container formats.

References

[1] M. Bartel, J. Boyer, B. Fox, B. LaMacchia and E. Simon, XML-Signature Syntax and Processing, World Wide Web Consortium, Cambridge, Massachusetts (www.w3.org/TR/xmldsig-core), 2009.

[2] D. Beckett and T. Berners-Lee, Turtle: Terse RDF Tripe Language, World Wide Web Consortium, Cambridge, Massachusetts (www.w3.org/TeamSubmission/turtle), 2008.

[3] T. Berners-Lee, R. Fielding and L. Masinter, Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396 (www.ietf.org/rfc/rfc2396.txt), 1998.

[4] B. Carrier, Defining digital forensic examination and analysis tools using abstraction layers, *International Journal of Digital Evidence*, vol. 1(4), 2003.

[5] J. Carroll, Signing RDF graphs, *Proceedings of the Second International Semantic Web Conference*, pp. 369–384, 2003.

[6] J. Carroll, C. Bizer, P. Hayes and P. Stickler, Named graphs, provenance and trust, *Proceedings of the Fourteenth International Conference on the World Wide Web*, pp. 613–622, 2005.

[7] J. Carroll and P. Stickler, TriX: RDF Triples in XML, Technical Report HPL-2003-268, HP Labs, Palo Alto, California (www.hpl.hp.com/techreports/2004/HPL-2004-56.pdf), 2004.

[8] M. Cohen, PyFlag: An advanced network forensic framework, *Digital Investigation*, vol. 5(S1), pp. S112–S120, 2008.

[9] M. Cohen, S. Garfinkel and B. Schatz, Extending the Advanced Forensic Format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow, *Digital Evidence*, vol. 6(S1), pp. S57–S68, 2009.

[10] S. Garfinkel, Providing cryptographic security and evidentiary chain-of-custody with the Advanced Forensic Format, library and tools, *International Journal of Digital Crime and Forensics*, vol. 1(1), pp. 1–28, 2009.

[11] S. Garfinkel D. Malan, K. Dubec, C. Stevens and C. Pham, Advanced Forensic Format: An open, extensible format for disk imaging, in *Advances in Digital Forensics II*, M. Olivier and S. Shenoi (Eds.), Springer, Boston, Massachusetts, pp. 13–27, 2006.

[12] R. Meijer, The Carve Path Zero Storage Library and Filesystem (ocfa.sourceforge.net/libcarvpath), 2006.

[13] R. Moats, URN Syntax, RFC 2141 (www.ietf.org/rfc/rfc2141.txt), 1997.

[14] B. Schatz and A. Clark, An information architecture for digital evidence integration, *Proceedings of the AusCERT Asia Pacific Information Technology Security Conference*, pp. 15–29, 2006.

[15] P. Turner, Unification of digital evidence from disparate sources (digital evidence bags), *Digital Investigation*, vol. 2(3), pp. 223–228, 2005.

[16] K. Watkins, M. McWhorte, J. Long and B. Hill, Teleporter: An analytically and forensically sound duplicate transfer system, *Digital Investigation*, vol. 6(S1), pp. S43–S47, 2009.

[17] World Wide Web Consortium, RDF/XML Syntax Specification (Revised), Cambridge, Massachusetts (www.w3.org/TR/REC-rdf-syntax), 2004.