Hash Based Disk Imaging Using AFF4

By

Michael Cohen and Bradley Schatz

Presented At

The Digital Forensic Research Conference

DFRWS 2010 USA Portland, OR (Aug 2nd - 4th)

DFRWS is dedicated to the sharing of knowledge and ideas about digital forensics research. Ever since it organized the first open workshop devoted to digital forensics in 2001, DFRWS continues to bring academics and practitioners together in an informal environment. As a non-profit, volunteer organization, DFRWS sponsors technical working groups, annual conferences and challenges to help drive the direction of research and development.

http://dfrws.org
Hash based disk imaging with AFF4

Dr Bradley Schatz
Adjunct Associate Professor, Queensland University of Technology
Director, Schatz Forensic Pty Ltd

* Dr Michael Cohen
Australian Federal Police
Motivation
Current container formats only address storage and representation of a fraction of case related information

- **Data content**: *Single Streams*, 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, etc)
- **Context**: Environment the hard drive existed in, Case related information
- **Behaviour**: Error codes related to bad sectors

- **Efficiency**: *Storage space minimising, Random access performance, IO Bandwidth*
- **Authentication**: *Cryptographic signing, Hash storage*
- **Privacy**: Encryption, Redaction
- **Resilience**: Tolerance of underlying storage medium failure
Current tool interoperability is hampered by current container formats

- Too much copying
- Not enough accessible information
Foundations
AFF4 is a new container format for storing digital evidence
The AFF4 Data Storage Model

Data Model

Information Model

Unique Naming

IO Efficient Data Storage

Seekable Compression

Hierarchical Segment Storage
data and information are stored within a hierarchical storage container
that storage container is a zip file
bitwise data is stored in an efficient seekable compressed form: the Stream
The AFF4 Naming Approach

- Data Model
- Information Model
- Unique Naming
- IO Efficient Data Storage
data and information are represented by surrogates, which are identified by a globally unique name

```
\information.turtle
\aff4%3A%2F%2F4f9b6891-8ad3-46c2-a237-932d2c8afbf6\00000000
\aff4%3A%2F%2F4f9b6891-8ad3-46c2-a237-932d2c8afbf6\00000000.idx
\aff4%3A%2F%2F4f9b6891-8ad3-46c2-a237-932d2c8afbf6\00000001
\aff4%3A%2F%2F4f9b6891-8ad3-46c2-a237-932d2c8afbf6\00000001.idx
\aff4%3A%2F%2F4f9b6891-8ad3-46c2-a237-932d2c8afbf6\00000002
\aff4%3A%2F%2F4f9b6891-8ad3-46c2-a237-932d2c8afbf6\00000002.idx
\aff4%3A%2F%2F4f9b6891-8ad3-46c2-a237-932d2c8afbf6\00000003
\aff4%3A%2F%2F4f9b6891-8ad3-46c2-a237-932d2c8afbf6\00000003.idx
```
The AFF4 Data Model

Data referencing and composition

Data Model

Information Model

Unique Naming

IO Efficient Data Storage
Maps enable describing discontiguous byte ranges

0,0,aff4://da0d1948-846f-491d-8183-34ae691e8293
6144,0, http://libaff.org/2009/aff4#UnknownData
8192,6144, aff4://da0d1948-846f-491d-8183-34ae691e8293
20480,0, http://libaff.org/2009/aff4#ZeroData
22528,18432, aff4://da0d1948-846f-491d-8183-34ae691e8293
The AFF4 Information Model

- Data Model
- Information Model
- Unique Naming
- IO Efficient Data Storage

Representation of arbitrary information
“object” instances are uniquely identified by a globally unique naming scheme

- **Object 1**: aff4://500afaba-8224-4560-bfec-d911499d3a4e
  - Type: Stream
  - Size: 257949696
  - Hash: fa8904729b...
  - Description: “…”

- **Object 2**: aff4://ddb733ed-815c-4b97-afed-5b2ed5bc0e1f
  - Type: Tool
  - Name: aff4imager
  - Version: 0.1

- **Object 3**: aff4://ddefa98e-ccf4-4f23-99fb-3b95ab41dcd9
  - Type: Operator
  - Name: “Bradley Schatz”
  - Email: “…”
instances are serialized as RDF triples

```
aff4://500afaba-8224-4560-bfec-d911499d3a4e type Stream
aff4://500afaba-8224-4560-bfec-d911499d3a4e size "257949696"^^xsd:long
aff4://500afaba-8224-4560-bfec-d911499d3a4e hash "fa8904729b..."^^aff4:md5
aff4://500afaba-8224-4560-bfec-d911499d3a4e description "320GB WD HDD"

aff4://ddb733ed-815c-4b97-afd5-0b2ed5bc0e1f type Tool
aff4://ddb733ed-815c-4b97-afd5-0b2ed5bc0e1f name "aff4Imager"
aff4://ddb733ed-815c-4b97-afd5-0b2ed5bc0e1f version "0.1"

aff4://abec3efe-92b5-4142-baf5-6cdedebc561d type Operator
aff4://abec3efe-92b5-4142-baf5-6cdedebc561d name "Bradley Schatz"
aff4://abec3efe-92b5-4142-baf5-6cdedebc561d email bradley@foo.com
```
...and are stored within the "information.turtle" file using the Turtle RDF serialization.
properties of instances may be defined in any container

```plaintext
aff4://500afaba-8224-4560-bfec-d911499d3a4e aff4:type aff4:Stream
aff4://500afaba-8224-4560-bfec-d911499d3a4e aff4:size "257949696"^^xsd:long
aff4://500afaba-8224-4560-bfec-d911499d3a4e aff4:hash "fa8904729b.."^^aff4:md5
aff4://500afaba-8224-4560-bfec-d911499d3a4e aff4:description “320G WD HDD”
```

```plaintext
aff4://500afaba-8224-4560-bfec-d911499d3a4e aff4:hash “129b0afcc..”^^aff4:sha256
```
Current status
AFF4 Library

- Reference implementation of the AFF4 format
  - It helps us test and evolve the format by using it
  - Flexible – can combine all types of AFF4 objects together
  - Python bindings automatically generated
    - Easy to keep in sync with C library
    - Very Fast
  - Multithreaded
  - Easy to use
  - API still in flux
Libaff4 overview

- Single entry point to library is the Resolver class:
  - Persistent data store
  - Currently implemented using tdb the trivial database
    - Very fast key/value database
  - Object manipulation
    - Opening and creating new AFF4 objects
    - Loading new AFF4 volumes
  - Information model access
    - Resolving, setting and adding predicates and iterating over values.
  - Miscellaneous Registrations
    - Registering new RDF data types, aff4 object implementations, logging subsystem and a security provider for key management.
Example – Volume creation

1. Get resolver
   ```python
   oracle = pyaff4.Resolver()
   ```

2. Create a new volume
   ```python
   volume_urn = pyaff4.RDFURN()
   volume_urn.set("/tmp/test.zip")
   volume = oracle.create(pyaff4.AFF4_ZIP_VOLUME)
   oracle.set_value(volume.urn, pyaff4.AFF4_STORED, volume_urn)
   volume = volume.finish()
   volume_urn = volume.urn
   volume.cache_return()
   ```

3. Finish the volume
4. Return to cache

5. Create a new Stream stored In the new volume
   ```python
   image_fd = oracle.create(pyaff4.AFF4_IMAGE)
   oracle.set_value(image_fd.urn, pyaff4.AFF4_STORED, volume_urn)
   image_fd = image_fd.finish()
   ```

6. Finish the stream

7. Copy data Into the stream
   ```python
   fd = open("/bin/ls")
   while 1:
       data = fd.read(1000000)
       if not data: break
       image_fd.write(data)
   ```

8. Done, close everything
   ```python
   image_fd.close()
   volume = oracle.open(volume_urn, 'w')
   volume.close()
   ```

Note use of Open() to reacquire The volume lock
Typical use pattern

- **Creating a new object:**
  - Call `resolver.create(type)` to make a new object
    - It will receive a unique URI
  - Call `set()` method on it to set various predicates
    - Especially important is the `aff4:stored` predicate
  - When finished call the `finish()` method on it to complete the object
  - When done with the object call `close()` to finalize it.

- **Opening an existing object:**
  - Call `resolver.open(url, mode)` to get the object
  - Call the `close()` method when done.
Thread control and locking

- Since the library is multithreaded:
  - An object opened for writing will be locked to the calling thread.
    - The lock is not recursive so deadlocks can result
    - Library will raise an exception if a deadlock is detected.
  - Threads must release the object by calling the `cache_return()` method as soon as possible.
  - Threads can reacquire the same object by calling `resolver.open()` using its URL later.
    - Will block if object is locked by some other thread.
- No locks for reading
  - multiple simultaneous readers are allowed
Applications
Applications

- We can get some useful effects by combining AFF4 objects together:
  - Carving
    - Carver generates a sequence of maps from the original stream
    - Zero copy carving
  - RAID reassembly
    - Acquire each disk in a RAID as a separate Image stream
    - Build a map which provides access to the logical (Reassembled) view with no copy overheads.
Applications

- Sparse image support:
  - A map stream is placed in front of an Image stream
  - Data is written to the image stream, while the map is adjusted
  - Holes in the map may be represented as bytes taken from the aff4:zero special target
    - Similar to /dev/zero
  - Bad blocks can be represented as taken from the aff4:bad_block target.
    - This can be varied to represent different reasons for missing data
Applications

- TCP/IP stream reassembly
  - A reassembler creates a map which pieces together streams from TCP payloads.
  - Further protocol dissection can be applied to TCP streams
    - HTTP, POP, SMTP etc
- Zero copy process
- Maintain Provenance
This paper
Hash based imaging

- The concept:
  - Store references to byte streams (hashes) rather than actual bit sequences
  - Potential benefits: storage/bandwidth efficiency

- Previous work: Teleporter
  - Client server protocol for transmission and reconstruction of images
  - Hashes based on file content
  - Soundness with sparse and encrypted files
AFF4 Hash imager

- Break a hard disk image into a set of block runs.
  - e.g. Using filesystem block allocation information.
- Each block run is stored as an image stream
- Block run is addressed by hash
- The disk is represented as a map which targets all the block runs.
- Block runs can be omitted
  - Saving acquisition time and space
  - The same hash within the image (e.g. file copies).
  - The same hash within the corpus
The segmenting algorithm

- Tradeoffs
  - Maximum/minimum block size
  - Compressibility
  - Sequential IO
# Example map file

<table>
<thead>
<tr>
<th>Image offset</th>
<th>Target offset</th>
<th>Length (B)</th>
<th>Target stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8,192</td>
<td>aff4://2efe1ec75b170112b354b7e62a701f7929bff265</td>
</tr>
<tr>
<td>8,192</td>
<td>0</td>
<td>10,240,000</td>
<td>aff4://cdff2e763ed9b545b201b28780b4df9f45260619</td>
</tr>
<tr>
<td>10,248,192</td>
<td>0</td>
<td>10,240,000</td>
<td>aff4://f8a8f6f689a197d85905094b525ed2e9a9fa1e30a</td>
</tr>
<tr>
<td>20,488,192</td>
<td>0</td>
<td>10,240,000</td>
<td>aff4://c2bfe67cfb2e4cac7edbb3013888abea6a123a5</td>
</tr>
<tr>
<td>30,728,192</td>
<td>0</td>
<td>56,832</td>
<td>aff4://6ea4826afcfb22483ad02716027122b09cd77d6f</td>
</tr>
<tr>
<td>30,785,024</td>
<td>0</td>
<td>2,097,152</td>
<td>aff4://3ef9b3329d8b24f28824abc457781ab0a51e23e7</td>
</tr>
<tr>
<td>32,882,176</td>
<td>0</td>
<td>2,560</td>
<td>aff4://b49d7f48300701235231f6b6fc3d92a5630f9e70</td>
</tr>
<tr>
<td>32,884,736</td>
<td>0</td>
<td>4,096</td>
<td>aff4://01829df4c765b8fa8991537d641916ae368d71a5</td>
</tr>
<tr>
<td>32,888,832</td>
<td>0</td>
<td>908</td>
<td>aff4://e78424ed-9398-4c94-bc9f-e5e8684564ff/ntfs_image.dd/misc</td>
</tr>
<tr>
<td>32,889,740</td>
<td>1798</td>
<td>628</td>
<td>aff4://e78424ed-9398-4c94-bc9f-e5e8684564ff/ntfs_image.dd/misc</td>
</tr>
</tbody>
</table>
Results
Comparison of imaging times and volume size for various techniques

<table>
<thead>
<tr>
<th>Acquisition Method</th>
<th>Total Image Size (B)</th>
<th>Elapsed Time</th>
<th>User CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash based Imaging (No corpus)</td>
<td>1,586,315,782</td>
<td>7m12s</td>
<td>5m42s</td>
</tr>
<tr>
<td>Hash based Imaging (Full corpus)</td>
<td>108,573</td>
<td>4m41s</td>
<td>0m51s</td>
</tr>
<tr>
<td>EWF fast setting</td>
<td>1,622,220,104</td>
<td>7m11s</td>
<td>5m23s</td>
</tr>
<tr>
<td>AFFLIB Compression level 1 (fastest)</td>
<td>1,590,285,671</td>
<td>9m55s</td>
<td>7m7s</td>
</tr>
<tr>
<td>AFF4 imager</td>
<td>1,621,922,977</td>
<td>4m37s</td>
<td>5m43s</td>
</tr>
</tbody>
</table>
Compression is expensive

<table>
<thead>
<tr>
<th>Acquisition Method</th>
<th>Total Image Size (B)</th>
<th>Elapsed Time</th>
<th>User CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash based Imaging (No corpus)</td>
<td>1,586,315,782</td>
<td>7m12s</td>
<td>5m42s</td>
</tr>
<tr>
<td>Hash based Imaging (Full corpus)</td>
<td>108,573</td>
<td>4m41s</td>
<td>0m51s</td>
</tr>
<tr>
<td>EWF fast setting</td>
<td>1,622,220,104</td>
<td>7m11s</td>
<td>5m23s</td>
</tr>
<tr>
<td>AFFLIB Compression level 1 (fastest)</td>
<td>1,590,285,671</td>
<td>9m55s</td>
<td>7m7s</td>
</tr>
<tr>
<td>AFF4 imager</td>
<td>1,621,922,977</td>
<td>4m37s</td>
<td>5m43s</td>
</tr>
</tbody>
</table>
Hash based imaging may result in substantial space savings

<table>
<thead>
<tr>
<th>Acquisition Method</th>
<th>Total Image Size (B)</th>
<th>Elapsed Time</th>
<th>User CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash based Imaging (No corpus)</td>
<td>1,586,315,782</td>
<td>7m12s</td>
<td>5m42s</td>
</tr>
<tr>
<td>Hash based Imaging (Full corpus)</td>
<td>108,573</td>
<td>4m41s</td>
<td>0m51s</td>
</tr>
<tr>
<td>EWF fast setting</td>
<td>1,622,220,104</td>
<td>7m11s</td>
<td>5m23s</td>
</tr>
<tr>
<td>AFFLIB Compression level 1 (fastest)</td>
<td>1,590,285,671</td>
<td>9m55s</td>
<td>7m7s</td>
</tr>
<tr>
<td>AFF4 imager</td>
<td>1,621,922,977</td>
<td>4m37s</td>
<td>5m43s</td>
</tr>
</tbody>
</table>
Hash based imaging currently costs more time when used in absence of corpus

<table>
<thead>
<tr>
<th>Acquisition Method</th>
<th>Total Image Size (B)</th>
<th>Elapsed Time</th>
<th>User CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash based Imaging (No corpus)</td>
<td>1,586,315,782</td>
<td>7m12s</td>
<td>5m42s</td>
</tr>
<tr>
<td>Hash based Imaging (Full corpus)</td>
<td>108,573</td>
<td>4m41s</td>
<td>0m51s</td>
</tr>
<tr>
<td>EWF fast setting</td>
<td>1,622,220,104</td>
<td>7m11s</td>
<td>5m23s</td>
</tr>
<tr>
<td>AFFLIB Compression level 1 (fastest)</td>
<td>1,590,285,671</td>
<td>9m55s</td>
<td>7m7s</td>
</tr>
<tr>
<td>AFF4 imager</td>
<td>1,621,922,977</td>
<td><strong>4m37s</strong></td>
<td><strong>5m43s</strong></td>
</tr>
</tbody>
</table>
Conclusions
Conclusions

- Hash based imaging:
  - A space/bandwidth efficient approach to storing forensic images
  - Competitive with current compressed imaging speeds
Conclusions

- AFF4
  - Container format is rich and expressive
    - We did not discuss encryption and signing support
    - Can express any information using ours or third party defined vocabulary
    - Ideal for metadata interchange (it's basically XML)
  - AFF4 is extensible
    - Can register your own:
      - AFF4 object
      - RDF Datatype
    - Implementations may not implement all features
      - Nice for simple, embedded applications, can implement a very minimalistic AFF4 writer
Future

• AFF4 aware imagers
  – Shift more analytic building blocks to acquisition stage

• Formal information models

• AFF4 aware tools
  – Support modular and composeable tools
  – Integration of arbitrary information
The AFF4 format & API will stabilise to beta shortly

• Parallel implementations
  – C and Java
  – Python binding to C

• Integration with
  – PyFlag
  – Sleuthkit

• For more information
Thank you

Dr Bradley Schatz
b.schatz@qut.edu.au