Wirespeed: Extending The Aff4 Container Format For Scalable Acquisition And Live Analysis

By
Bradley Schatz

Presented At
The Digital Forensic Research Conference
DFRWS 2015 USA Philadelphia, PA (Aug 9th - 13th)

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
Wirespeed:
Extending the AFF4 forensic container format for scalable acquisition and live analysis

Dr. Bradley Schatz
Director, Schatz Forensic

DFRWS Conference 2015
© Schatz Forensic 2015
Overview

• The current approach to forensic acquisition is a bottleneck in the forensic process

• Propose additions to the AFF4 container format to support:
  – Partial acquisition
  – Acquisition at maximal I/O rates

• Empirical results of the proposed approach
Background
Pick one of the below
You can’t have both

- Near immediate results at the expense of potentially missing evidence
- Live forensics
- Triage
- Physical Acquisition

You preserve everything but analysis will have to wait
How can we reduce latency? While maximising completeness

- Increase I/O throughput?
- Live analysis while we acquire?
- Acquire/Analyse by priority?

Dynamic partial acquisition?

Physical Acquisition

Live forensics

Triage
Why can’t I have both?

The de-facto standard evidence containers are a limiting factor.

- Linear complete bit stream
  - All or nothing preservation choice
  - Prevents non-linear/prioritised preservation of evidence
- Heavyweight compression (Inflate)
  - Limiting factor on current CPU’s (even with multi-core threading)
- Linear bytestream hash
  - Prevents non-linear/prioritised preservation
  - Hashing is limiting factor at high bitrates and with low CPU resources
- Single storage device
  - Evidence output device I/O rate often limiting factor
- Logical imaging
  - Missing raw filesystem and volume metadata
I/O Throughput
Is there actually a problem with throughput here?

- Research publications
  - FastDD $\leq 110$ MB/s [Bertasi & Zago 2013]

- Practitioner reports
  - Low 100’s MB/s [Zimmerman 2013]

- Vendor marketing
  - Hardware devices promising 250MB/s [Tableau 2014]
I/O throughput in acquisition is a systems problem

<table>
<thead>
<tr>
<th>Target Storage</th>
<th>Max Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current generation 3.5” 7200rpm SATA</td>
<td>200 MB/s</td>
</tr>
<tr>
<td>Intel 730 SSD</td>
<td>550 MB/s</td>
</tr>
<tr>
<td>Macbook Pro 1TB (real data)</td>
<td>100 MB/s</td>
</tr>
<tr>
<td>Macbook Pro 1TB (sparse)</td>
<td>1 GB/s</td>
</tr>
<tr>
<td>RAID 15000rpm SAS</td>
<td>&gt; 1 GB/s</td>
</tr>
</tbody>
</table>
Inflate compression is costly in CPU resources

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Throughput MB/s*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inflate</td>
<td>39.42</td>
</tr>
<tr>
<td>Snappy (Google BigTable/MapReduce)</td>
<td>1,405.42</td>
</tr>
<tr>
<td>LZO (ZFS)</td>
<td>1,538.31</td>
</tr>
</tbody>
</table>

*Single core of quad core i7-4770 3.4Ghz
Hashing is the next most expensive acquisition operation

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Throughput MB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA1</td>
<td>619.23</td>
</tr>
<tr>
<td>MD5</td>
<td>745.65</td>
</tr>
<tr>
<td>Blake2b</td>
<td>601.87</td>
</tr>
</tbody>
</table>

*Single core of quad core i7-4770 3.4Ghz*
I/O Rate of acquisition is a systems problem

<table>
<thead>
<tr>
<th>Output</th>
<th>Gb/s</th>
<th>MB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>SATA3</td>
<td>6</td>
<td>600</td>
</tr>
<tr>
<td>USB3</td>
<td>5</td>
<td>500</td>
</tr>
<tr>
<td>Commodity SATA 7200 rpm</td>
<td></td>
<td>200</td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>USB2</td>
<td>.48</td>
<td>48</td>
</tr>
</tbody>
</table>
Our proposal
Extensions to AFF4
AFF4 in a nutshell

- Virtual block storage (Maps)
  - Non-linear, composable
- Compressed block storage (Streams)
- Globally unique naming scheme
- There is an object representing each entity

Zero filled storage (aff4:Zero)

Image (Map)

Compressed block storage

/mapID/map/mapID/index
/streamID/00000
/streamID/00000/index
/streamID/00001
/streamID/00001/index
...
Faster compression
Symbolic sections

• Extension: we define virtual bytestreams “aff4:SymbolicStream00” to “aff4:SymbolicStreamFF”

• Synonyms aff4:Zero and aff4:FF
  – Use case: Zero filled sectors and erased flash blocks

AFF4 Map example

0,0,aff4://0466b8fb-9af0-4ef2-b36c-8b0d90fc0ac2>
4096,0,aff4:SymbolicStreamFF
8192,4096, aff4://0466b8fb-9af0-4ef2-b36c-8b0d90fc0ac2>
Partial acquisition

- Challenge: Representing what we didn’t acquire
- Extensions: we define two symbols
  - aff4:UnreadableData : Blocks that we tried to read but couldn’t
  - aff4:UnknownData : Blocks that we never even tried to read

AFF4 Map example

```
0,0,aff4://0466b8fb-9af0-4ef2-b36c-8b0d90fc0ac2>
4096,0,aff4:UnknownData
8192,4096, aff4://0466b8fb-9af0-4ef2-b36c-8b0d90fc0ac2>
```
Faster compression
More speed-efficient algorithms

• Extension: the storage stream now has property called “aff4:compressionMethod”

AFF4 Stream example

```xml
<aff4://0466b8fb-9af0-4ef2-b36c-8b0d90fc0ac2> a aff4:stream ;
    aff4:CompressionMethod <http://code.google.com/p/snappy/> ;
    aff4:chunk_size "32768"^^xsd:int ;
    aff4:size "294912"^^xsd:long ;
```
Faster hashing
Non-linear parallelised block hashing

• Our proposal:
  – Deprecate the linear bytestream hash
  – Parallelise hashing by using segment hashes
  – Hashing symbolic chunks is a waste of CPU resources – hash the map instead
  – Take a singular hash of the above hashes
Faster hashing
Non-linear parallelised block hashing

Image (Map)
mapPointHash = sha256(mapID/map)
mapIndexHash = sha256(mapID/index)

Stream B
blockHashesHash = sha256(h₁, h₀, h₄, h₂, ...)

Stream B
block hashes

blockHash = sha256(blockHashesHash₀ .. blockHashesHashₙ . mapPointHash . mapIndexHash)
Aggregate Output Channels

• Use the aggregate I/O capacity of the device
Experimental validation
Methodology

• We built a forensic acquisition/analysis system
  – Non-linear, partial acquisition & live analysis (called Wirespeed)
• Prepare testbed
  – Target disk: Intel 730 240G SSD (max read 530 MB/s)
  – Destination disks: Toshiba 2TB 7200RPM SATA (max write near 200 MB/s)
  – Computer 1: 4 core i7-4770R 3.20GHz
  – Computer 2: 2 core i5-3337U 1.80GHz
• Prepare test sample
• Test, varying on
  – CPU
  – IO Channels
Test standard composition
Stored block size –v- LBA address

Compressed chunk size

- Windows 8.1: 10.2G
- /dev/random: 38.4G
- Govdocs1 (1-75,1-40): 59.8G
- Empty space (zeros)
Compression is faster than raw
Single output drive

Compression vs No Compression

High entropy data write I/O limited

Low entropy data read throughput limited

Medium entropy data throughput exceeds max output throughput
Multiple output channels increases throughput
Especially for uncompressed data.

High entropy data
Block based hashing beats linear stream hashing with low powered multicore CPU’s

Dual core i5

Stream Hash vs Block Hash (MB/s)

Sparse data
Read I/O limited

Sparse data Max CPU hash throughput
Non-linear partial imaging gives significant gains over linear
The proposed approach gives significant throughput gains over current implementations.

<table>
<thead>
<tr>
<th>Acquisition application</th>
<th>I7-4770R 3.2 GHz system</th>
<th>I5-3337U 1.8GHz system</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTK Imager</td>
<td>20:10 (198 MB/s)</td>
<td>37:38 (106MB/s)</td>
</tr>
<tr>
<td>X-Ways Forensics</td>
<td>13:58 (286 MB/s)</td>
<td>33:23 (120 MB/s)</td>
</tr>
<tr>
<td>Wirespeed (linear)</td>
<td>11:29 (384 MB/s)</td>
<td>15:08 (264 MB/s)</td>
</tr>
</tbody>
</table>
## Comparative acquisition speeds

<table>
<thead>
<tr>
<th></th>
<th>1 Stripe</th>
<th>2 Stripes</th>
<th>3 Stripes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wirespeed</strong></td>
<td>11:29 (384 MB/s)</td>
<td>8:00 (500 MB/s)</td>
<td>7:30 (533 MB/s)</td>
</tr>
<tr>
<td>(linear)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FTK Imager</strong></td>
<td>20:10 (198 MB/s)</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>X-Ways</strong></td>
<td>13:58 (286 MB/s)</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Forensics</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Wirespeed</strong></td>
<td>8:21 (229 MB/s)</td>
<td>4:42 (408 MB/s)</td>
<td>4:17 (447 MB/s)</td>
</tr>
<tr>
<td>(allocated)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

• Existing image formats are a limitation
  – Linear byte stream hash
  – Inflate algorithm

• Extensions to the AFF4 format proposed
  – Faster hashing and compression
  – Partial images
  – Exploitation of aggregate IO channels

• Proof of concept demonstrated significant throughput gains and improved latency

• Our implementation is available at:
  – http://wirespeed.io