On Efficiency of Artifact Lookup Strategies in Digital Forensics

L. Liebler, P. Schmitt, H. Baier, F. Breitinger

Hochschule Darmstadt, CRISP, da/sec Security Group

2019-04-25
Agenda

Motivation

Candidates

Requirements / Capabilities

Extensions to hbft and fhmap

Evaluation

Conclusion
Motivation

Candidates

Requirements / Capabilities

Extensions to hbft and fhmap

Evaluation

Conclusion
Data overload

1 TiB digital text equals (approximately):

- 220 million printed pages: 1 page = 5000 characters.
- 1 million kg paper: printed one-sided.

Source: www.spiegel.de

Source: Eric Gaba, Wikimedia; CC-BY-SA
Motivation

Finding relevant artifacts resembles ...

Digital forensic experts need **automated** filtering to reduce the haystack or increase the needle.
Motivation

General process pipeline: approximate matching

1. Construction phase of data set (e.g., a blacklist) using approximate matching:
   - Extract blocks / features
   - Hash them
   - Insert hashed block into ’database’
   - Sorting difficult due to fuzzy nature of input

2. Lookup phase:
   - Extract blocks / features from seized device
   - Hash them
   - Comparison against the ’database’

We focus on alternative ’database’ approaches to solve the database lookup problem.
Motivation

Use Case / Goals

1. Use case: find efficient (i.e. fast) strategies to detect known digital traces, e.g., in the context of
   ▶ white- and blacklisting scenarios in forensic use cases
   ▶ carving
   ▶ within large corpora (memory-, lookup-efficient)

2. General goal: discuss, reassess and extend three widespread lookup strategies

3. Further goals:
   ▶ deduplication (i.e., remove common blocks)
   ▶ adding and deleting items
Candidates

Motivation

Candidates

Requirements / Capabilities

Extensions to hbft and fhmap

Evaluation

Conclusion
Candidate preselection

Preselection of three 'database' approaches and corresponding lookup strategies suitable for storing hash-based fragments:

- **hashdb**: Hash-based carving due to Garfinkel et al. [GM15], part of the bulk_extractor
- **hbft**: Hierarchical Bloom filter trees originally due to Breitinger et al. [BRB14]
- **fhmap**: flat hash map, presented by Malte Skarupke at C++Now in 2018


Candidates

hashdb: main features

- Uses lightning memory mapped database structure (LMDB)
- Handles large data sets (1 million files in [GM15])
- Read-optimised (read-only transactions operate in parallel)
- Built-in deduplication (common block / multi hit prevention)
- Adding and deleting items is possible
- Uses fixed sliding window for block building
Bloom filter (Burton Howard Bloom in 1970)

- Very space-efficient + probabilistic data structure
- Array with the size of $m$ bits ($m = 18$ in the following sample Bloom filter)

Source: https://commons.wikimedia.org/wiki/User:David_Eppstein
Hierarchical Bloom filter tree (hbft): concept

Candidates

mrsh-hbft proof-of-concept by Lillis et al. [LBS17]

Candidates

**hbft: main features**

- Lookup complexity of $O(\log(n))$

- **False positive rate** of a bloom filter is influenced by three parameters:
  1. Size of the filter $m$
  2. Number of $n$ inserted elements of a set $S = \{s_1, ... s_n\}$
  3. Number of used hash functions $k$

- Deletion of elements hardly possible

---

L. Liebler, P. Schmitt, H. Baier, F. Breitinger
flat hash map (fhmap): main features

- Fast hash table (actually the author claims that the implementation features the fastest lookups until now): lookup complexity of $O(1)$
- Robin Hood hashing according to [CLM85]: ensures that most of the elements are close to their ideal entry in the table by rearrangement
- No false positives

Requirements / Capabilities

Motivation

Candidates

Requirements / Capabilities

Extensions to hbft and fhmap

Evaluation

Conclusion
Multi hit handling

- Identical blocks of **different** files (e.g., file header structures, statically linked libraries)
- Often no value to an analyst (block is not characteristic for a given artifact)
- Needs to be filtered out (during construction or lookup phase)
- Keep multi hits which only appear within one file
Summary capability analysis

A direct comparison is hard as capabilities differ → re-implementation of several features needed

<table>
<thead>
<tr>
<th></th>
<th>hashdb</th>
<th>hbft</th>
<th>fhtmap</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storing Technique</strong></td>
<td>LMDB</td>
<td>Bloom filter tree</td>
<td>Hash table</td>
</tr>
<tr>
<td><strong>Block Building</strong></td>
<td>Fixed sliding window</td>
<td><strong>Fixed size</strong>*/ rolling hash</td>
<td><strong>Fixed size</strong>*/ rolling hash*</td>
</tr>
<tr>
<td><strong>Block Hashing</strong></td>
<td>MD5</td>
<td>FNV-256</td>
<td>FNV-1</td>
</tr>
<tr>
<td><strong>Multithreading</strong></td>
<td>All phases</td>
<td><strong>Block building</strong>*</td>
<td><strong>Block building</strong>*</td>
</tr>
<tr>
<td><strong>Multihit Handling</strong></td>
<td>✓</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><strong>Add / Remove Hashes</strong></td>
<td>✓/✓</td>
<td>Partially / ✗</td>
<td>✓/✓</td>
</tr>
<tr>
<td><strong>Prefilter</strong></td>
<td>”Hash Store”</td>
<td>Root Bloom filter</td>
<td>✗</td>
</tr>
<tr>
<td><strong>False Positives</strong></td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Storing Type</strong></td>
<td>Single-level storage</td>
<td>Primary storage</td>
<td>Primary storage</td>
</tr>
<tr>
<td><strong>Not limited to RAM</strong></td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Persistent Database</strong></td>
<td>✓</td>
<td>✓</td>
<td>*</td>
</tr>
</tbody>
</table>
Extensions to hbft and fhmap

Motivation

Candidates

Requirements / Capabilities

Extensions to hbft and fhmap

Evaluation

Conclusion
Overview of implemented extensions

- Multi hit prevention hbft:
  - Tree-filter based
  - Global-filter based
  - Evaluation

- Multi hit prevention fhmap

- Parallelisation of block building
Parallelisation of block building

<table>
<thead>
<tr>
<th></th>
<th>Singlethread</th>
<th>Multithread (8 Threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real</td>
<td>43.82 s</td>
<td>13.59 s</td>
</tr>
<tr>
<td>CPU</td>
<td>35.87 s</td>
<td>49.25 s</td>
</tr>
</tbody>
</table>
Evaluation

Motivation

Candidates

Requirements / Capabilities

Extensions to hbft and fhmap

Evaluation

Conclusion
Overview of evaluated aspects

- Memory consumption
- Run time of construction phase:
  - Single threaded
  - Multiple threaded
- Run time of deduplication:
  - Single threaded
  - Multiple threaded
- Run time of lookup phase (depending on matching rate)
Evaluation

Lookup evaluation

- hbft fi
- hbft ro
- fhmap fi
- fhmap ro
- hashdb

a) Single thread
b) Single thread
c) Multiple threads (8)
## Overall evaluation

<table>
<thead>
<tr>
<th>Feature</th>
<th>hashdb</th>
<th>hbft</th>
<th>fhmap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multithreading</td>
<td>++</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Add Hashes</td>
<td>++</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>Remove Hashes</td>
<td>++</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>Limited to RAM</td>
<td>++</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Transactions</td>
<td>++</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Persistent Database</td>
<td>++</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Prefilter</td>
<td>+</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>False Positives</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Build Phase (Single)</td>
<td>-</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Build Phase (Multiple)</td>
<td>+</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Deduplication Phase (Single)</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Deduplication Phase (Multiple)</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Lookup Phase (Single)</td>
<td>-</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Lookup Phase (Multiple)</td>
<td>0</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>
Conclusion

Motivation

Candidates

Requirements / Capabilities

Extensions to hbft and fhmap

Evaluation

Conclusion
fhmap outperforms both hbft and hashdb for our use case
Extending hbft is hard without loosing its advantages
fhmap integrated into the memory carving engine
Contact

▶ harald.baier@h-da.de / lorenz.liebler@h-da.de / FBreitinger@newhaven.edu

▶ Interested in internship at CRISP?