Content Triage with Similarity Digests
- The M57 Case Study

By
Vassil Roussev and Candice Quates

Presented At
The Digital Forensic Research Conference
DFRWS 2012 USA  Washington, DC (Aug 6th - 8th)

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
Content triage with similarity digests: The M57 case study

Vassil Roussev
Candice Quates
The M57 Case Study

Introduction
M57: The company & setup

- **Employees:**
  - President: Pat McGoo
  - IT: Terry
  - Researchers: Jo, Charlie

- **Period**
  - 11/20/2009 Jo’s computer replaced
  - Last day: police kick down the door

- **Data**
  - Daily HDD, RAM, network captures
M57: The data (1.5 TB)

- HDD images
  - 84 images, 10-40GB each
  - Total: **1,423 GB**

- RAM snapshots
  - 78 snapshots, 256-1024 MB each
  - Total: **107 GB**

- Network:
  - 49 traces, 4.6 GB

- USB
  - 4.1 GB

- Kitty set
  - 125 JPEGs, 224 MB
Scenario #1: Contraband

Setup:
- From the detective reports in the scenario, there is reason to suspect that one of M57's computers (Jo’s) has been used in the contraband of "kitty porn".

Questions:
- Were any M57 computers used in contraband?
- If so, when did the accident happen?
- Is there evidence of intent?
- How was the content distributed?
- Was any of the content sent outside the company network?
Scenario #2: Eavesdropping

- **Setup:**
  - It is suspected that somebody is spying on the CEO (Pat) electronically.

- **Plan?**
  - Search for potentially rogue processes that might have been introduced on his computer.
  - First HDD image is clean and serves as baseline.
Scenario #3: Corporate espionage

- Setup:
  - There is suspicion that somebody has leaked company secrets.

- Plan?
  - Search RAM snapshots for interesting processes
The need for better triage
Triage

- **Fast, reliable** initial screen of the acquired data:
  - *fast*: all you can do in 5/10/15/ ... min;
  - *reliable*: provides *strong hints* (low FP).

- **Goals:**
  - Identify the most (ir)relevant targets/artifacts;
  - Build an overall understanding of the case—what are the likely answers?

- **Location of work:**
  - We assume post-acquisition work in a lab, but
  - It could be done in the field (given enough hardware)
Metadata- vs content-based analysis

 Metadata-based analysis
  o Use FS metadata, registry, logs, etc.
  o **Pro**: small volume, high-level logical information
  o **Con**: not looking at the data, cannot see remnants, does not work on a data dump (e.g. RAM), metadata is fragile
  ➔ Typical basis for (manual) triage

 Content analysis
  o **Works on actual data content**
    ▪ File/block hashes, indexing, carving, etc.
  o **Pro**: looking at actual data, can work with pieces
  o **Con**: large volume, lower level data
  ➔ Almost never used in triage (perceived as too slow)
Why is content analysis so slow?

- Forensic Target (1.5TB)
  - Clone @150MB/s ~3 hrs
  - Process @10MB/s ~42 hrs

► We can start working on the case after 42 hours (!)
Why is content analysis so slow?

Forensic Target (1.5TB)

Clone @150MB/s ~3 hrs

Process @10MB/s ~42 hrs

- carving
- indexing
- file hashing
Why is content analysis so slow?

Forensic Target (1.5TB)

Clone
@150MB/s
~3 hrs

Process
@10MB/s
~42 hrs

How do we bypass these to enable content triage?

carving
indexing
file hashing
Data Correlation
with similarity digests
Motivation for similarity approach: *Traditional hash filtering is failing*

- **Known file filtering:**
  - Crypto-hash known files, store in library (e.g. NSRL)
  - Hash files on target
  - Filter in/out depending on interest

- **Challenges**
  - Static libraries are falling behind
    - Dynamic software updates, trivial artifact transformations
  - We need *version* correlation
  - Need to find embedded objects
    - Block/file in file/volume/network trace
  - Need higher-level correlations
    - Disk-to-RAM
    - Disk-to-network
Scenario #1: fragment identification

Given a fragment, identify source

- **Minimum** fragments of interest are 1-4KB in size
- Fragment *alignment is arbitrary*
Scenario #2: artifact similarity

- Given two binary objects, detect similarity/versioning:
  - Similarity here is purely syntactic;
  - Relies on commonality of the binary representations.

Similar files (shared content/format)

Similar drives (shared blocks/files)
Common solution: similarity digests

Is this fragment present on the drive?  \(0 \ldots 100\)

Are these artifacts correlated?  \(0 \ldots 100\)

All correlations based on bitstream commonality
The M57 Case Study

Using sdhash for triage
sdhash-2.2 generation rates

- sdhash generation is I/O-bound
- it can be run in line with imaging
### sdhash generation times (M57)

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Size (GB)</th>
<th>Time (min)</th>
<th>Rate (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDD</td>
<td>1,423.0</td>
<td>168.00</td>
<td>143</td>
</tr>
<tr>
<td>RAM</td>
<td>107.0</td>
<td>10.70</td>
<td>166</td>
</tr>
<tr>
<td>Network</td>
<td>4.6</td>
<td>0.40</td>
<td>196</td>
</tr>
<tr>
<td>USB disk</td>
<td>4.1</td>
<td>0.45</td>
<td>155</td>
</tr>
<tr>
<td>Kitty</td>
<td>0.2</td>
<td>0.08</td>
<td>45</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1,538.9</strong></td>
<td><strong>179.63</strong></td>
<td><strong>143</strong></td>
</tr>
</tbody>
</table>

- Dell PowerEdge R710 server
  - 2 x Intel Xeon CPUs @2.93GHz six-core with H/T 12(24) threads
  - 72GiB of RAM @800MHz
Scenario #1: Contraband

Setup:
- From the detective reports in the scenario, there is reason to suspect that one of M57's computers (Jo's) has been used in the contraband of "kitty porn".

Questions:
- Were any M57 computers used in contraband?
- If so, when did the accident happen?
- Is there evidence of intent?
- How was the content distributed?
- Was any of the content sent outside the company network?
Query 1: Search Jo’s HDD for kitty images

Jo’s computer: Number of instances found by date

260GB → 55 min → 123 sec
Query 2: What processes were running?

- Search Jo’s RAM for traces of installed executables

<table>
<thead>
<tr>
<th>Date</th>
<th>Directory Path</th>
<th>Process Name</th>
<th>Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/03</td>
<td>.../Downloads/TrueCrypt Setup 6.3a.exe</td>
<td>TrueCrypt Setup.exe</td>
<td>092</td>
</tr>
<tr>
<td></td>
<td>.../TrueCrypt Format.exe</td>
<td>TrueCrypt Format.exe</td>
<td>090</td>
</tr>
<tr>
<td></td>
<td>.../TrueCrypt Setup.exe</td>
<td>TrueCrypt Setup.exe</td>
<td>092</td>
</tr>
<tr>
<td></td>
<td>.../TrueCrypt.exe</td>
<td>TrueCrypt.exe</td>
<td>092</td>
</tr>
<tr>
<td>12/04</td>
<td>.../Downloads/TrueCrypt Setup 6.3a.exe</td>
<td>TrueCrypt Setup.exe</td>
<td>063</td>
</tr>
<tr>
<td></td>
<td>.../TrueCrypt Setup.exe</td>
<td>TrueCrypt Setup.exe</td>
<td>063</td>
</tr>
<tr>
<td>12/09</td>
<td>.../Downloads/TrueCrypt Setup 6.3a.exe</td>
<td>TrueCrypt Setup.exe</td>
<td>084</td>
</tr>
<tr>
<td></td>
<td>.../TrueCrypt Format.exe</td>
<td>TrueCrypt Format.exe</td>
<td>079</td>
</tr>
<tr>
<td></td>
<td>.../TrueCrypt Setup.exe</td>
<td>TrueCrypt Setup.exe</td>
<td>084</td>
</tr>
<tr>
<td></td>
<td>.../TrueCrypt.exe</td>
<td>TrueCrypt.exe</td>
<td>090</td>
</tr>
<tr>
<td>12/10</td>
<td>.../TrueCrypt.exe</td>
<td>TrueCrypt.exe</td>
<td>092</td>
</tr>
<tr>
<td>12/11 - pre-raid</td>
<td>.../TrueCrypt Format.exe</td>
<td>TrueCrypt Format.exe</td>
<td>086</td>
</tr>
<tr>
<td></td>
<td>.../TrueCrypt.exe</td>
<td>TrueCrypt.exe</td>
<td>079</td>
</tr>
</tbody>
</table>
Scenario #2: Eavesdropping

- **Setup:**
  - It is suspected that somebody is spying on the CEO (Pat) electronically.

- **Plan?**
  - Search for potentially rogue processes that might have been introduced on his computer.
  - First HDD image is clean and serves as baseline.
Eavesdropping timeline

11/16, [71] not in baseline
   Present: Java, Firefox, python, mdd_1.3.exe.

11/19, [95] not in baseline
   Acrobat Reader 9 installed or updated, including Adobe Air.
   18 other programs from 11/16 still present.

11/20, [289]
   Windows Update run: many new dlls in the _restore and SoftwareDistribution folders.

11/23, [561]
   Windows Update has run

11/30, [274]
   Likely a Brother printer driver installed.
   Acrobat/Firefox still present.

12/03, [649]
   AVG has been updated.
   XP Advanced Keylogger appears:
   XP Advanced/DLLs/ToolKeyloggerDLL.dll 087
   XP Advanced/SkinMagic.dll            027
   XP Advanced/ToolKeylogger.exe        024

12/07, [460]
   More Brother printer related files.
   InstallShield leftovers present.
   win32dd present.
   XP Advanced Keylogger is no longer here.
   RealVNC VNC4 has been installed and run:
   RealVNC/VNC4/logmessages.dll          068
   RealVNC/VNC4/winvnc4.exe              046
   RealVNC/VNC4/wm_hooks.dll             023

12/10, [1240]
   AVG updated.
   IE8 and Windows updated.
   VNC still present.

12/11, [634]
   VNC present.
Scenario #3: Corporate espionage

Setup:
- There is suspicion that somebody has leaked company secrets.

Plan?
- Search RAM snapshots for interesting processes
Scenario #3: Findings

- **RAM**
  - "Cygnus FREE EDITION" hex editor
    - On 11/24, 11/30, 12/02, 12/03, and 12/10;
  - "Invisible Secrets 2.1"
    - blowfish.dll, jpgcarrier.dll, bmpcarrier.dll
    ➔ likely stego tool

- **USB**
  - insecr2.exe
  - /microscope.jpg
  - /microscope1.jpg
  - /astronaut.jpg
  - /astronaut1.jpg
  - /Email/Charlie_..._Sent_astronaut1.jpg
  - /Email/other/Charlie_..._Sent_microscope1.jpg
M57 Conclusions

- Using sdhash, we can outline the solution of all three cases in about **120 min** of extra processing.
  - This assumes HDD/RAM hash generation while cloning.
  - This could be further improved by running the queries in R/T in parallel with acquisition.

- The tool enables differential analysis that is simple, fast, robust, and generic.
  - Most processing can run in parallel with acquisition.
  - In effect, it can replace carving/indexing during triage.
  - It does not require much expertise to apply; results are intuitive.
  - The analysis can be highly automated; higher-level analysis can be built on top.
Development Status
Architecture

C++ Client: sdhash-cli

Web GUI (python)

Server: sdhash-srv

Custom clients (20+ languages)

Apache Thrift C/S Protocol

Python

Other

SWIG-based APIs

Cross-platform C++ API: libsdbf

Third-party C++ libraries: boost, thrift, openssl (thrust, TBB)
Availability

- sdhash.org
  - Source
  - Windows exe
    - 32-/64-bit executables
  - Linux
    - rpm/deb packages
  - API documentation
  - Repository
  - Papers/presentations
**sdhash-2.2 comparison performance**

- **Small file comparison (1 core, Intel X5670)**
  - 10KB vs. 10KB 0.0061 ms
  - 100KB vs. 100KB 0.0125 ms
  - 1MB vs. 1MB 0.4300 ms
  - 10MB vs. 10MB 41.0000 ms

- **Large file/streaming comparison (12 cores) in seconds**

<table>
<thead>
<tr>
<th></th>
<th>100MB</th>
<th>125MB</th>
<th>150MB</th>
<th>200MB</th>
<th>500MB</th>
<th>1000MB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>100MB</strong></td>
<td>0.76</td>
<td>0.93</td>
<td>1.00</td>
<td>1.36</td>
<td>3.53</td>
<td>6.61</td>
</tr>
<tr>
<td><strong>125MB</strong></td>
<td>0.93</td>
<td>0.96</td>
<td>1.30</td>
<td>1.84</td>
<td>4.10</td>
<td>8.60</td>
</tr>
<tr>
<td><strong>150MB</strong></td>
<td>1.00</td>
<td>1.30</td>
<td>1.58</td>
<td>2.28</td>
<td>5.33</td>
<td>10.30</td>
</tr>
<tr>
<td><strong>200MB</strong></td>
<td>1.36</td>
<td>1.84</td>
<td>2.28</td>
<td>3.00</td>
<td>7.10</td>
<td>13.80</td>
</tr>
</tbody>
</table>
Todo: Scaling up to NSRL

- **Goal:**
  - Maintain R/T performance (100-150 MB/s) with 1TB reference set.

- **Approach:**
  - Pre-filtering/indexing using extra Bloom filters

- **Estimated cost:**
  - Approximately 2.5% extra; i.e. increase from 2.5 to 5% of reference data
  - 50GB per TB of data
  - Requires RAM-optimized server (e.g. 256GB → ~$7k)
Scaling up to NSRL (2)

Aggregate filters (index)

sdbf hashes, 100MB each
Scaling up to NSRL (2)

Aggregate filters (index)

sha1

sdbf hashes, 100MB each
Todo list

- **libsdbf**
  - Rewrite parallelization using *thrust, tbb, thrift*, or similar
  - Implement pre-filtering/indexing
  - GPU acceleration

- **sdhash**
  - More command line options/compatibility w/ssdeep
  - Pcap front end
    - payload extraction, file discovery, time-lining

- **sdhash-srv/sdhash-cli**
  - Multi-server deployment
  - GUI
Further Development

- Integration w/ RDS
  - `sdhash-set`: construct SDBFs from existing SHA1 sets
    - Compare/identify whole folders, distributions, etc.

- Structural feature selection
  - E.g., exe/dll, pdf, zip, ...

- Optimizations
  - Skipping
    - Under `min` continuous block assumption
  - Cluster “core” extraction/comparison

- Representation
  - Multi-resolution digests
  - New crypto hashes
  - Data offsets
Thank you!

- [http://sdhash.org](http://sdhash.org)

- sdhash tutorial: Wed, Aug 8 @3pm

- Vassil Roussev
  [vassil@roussev.net](mailto:vassil@roussev.net)
Similarity digests

Overview
Generating **sdhash** fingerprints (1)

**Digital artifact**

(block/file/packet/volume) as byte stream

**Features**

(all 64-byte sequences)
Generating *sdhash* fingerprints (2)

Digital artifact

Select characteristic features
(statistically improbable/rare)
Generating sdhash fingerprints (3)

Feature Selection Process

All features

Rare Local Feature Selector

$H_{\text{norm}} \rightarrow 0.1000$

Weak Feature Filter

Data with low information content

$H_{\text{norm}} \rightarrow \text{doc files}$
Generating sdbf fingerprints (4)

Sequence of Bloom filters (sdbf)

- Generating sdbf fingerprints
- SHA-1
- Bloom filter
  - local SD fingerprint
  - 256 bytes
  - up to 128/160 features
- Artifact SD fingerprint

8-10K avg
Bloom filter (BF) comparison

Based on BF theory, overlap due to chance is analytically predictable.

Additional BF overlap is proportional to overlap in features.

\[ BF_{\text{Score}} \text{ is tuned such that } BF_{\text{Score}}(A_{\text{random}}, B_{\text{random}}) = 0. \]
**SDBF fingerprint comparison**

<table>
<thead>
<tr>
<th>SD_A</th>
<th>SD_B</th>
<th>BF_A^1</th>
<th>BF_B^1</th>
<th>BF_Score(BF_A^1, BF_B^1)</th>
<th>max_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF_A^1</td>
<td>BF_B^1</td>
<td>BF_Score(BF_A^1, BF_B^2)</td>
<td>max_2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF_A^2</td>
<td>BF_B^1</td>
<td>BF_Score(BF_A^2, BF_B^2)</td>
<td>max_2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF_A^n</td>
<td>BF_B^1</td>
<td>BF_Score(BF_A^n, BF_B^2)</td>
<td>max_n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF_A^1</td>
<td>BF_B^2</td>
<td>BF_Score(BF_A^1, BF_B^2)</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF_A^2</td>
<td>BF_B^2</td>
<td>BF_Score(BF_A^2, BF_B^2)</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF_A^n</td>
<td>BF_B^2</td>
<td>BF_Score(BF_A^n, BF_B^2)</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ SD_{Score}(A,B) = \text{Average}(\text{max}_1, \text{max}_2, ..., \text{max}_n) \]
Scaling up:
Block-aligned digests & parallelization
Block-aligned similarity digests ($sdbf-dd$)

Sequence of Bloom filters ($sdbf-dd$)

- $bf_1$
- $bf_2$
- $bf_3$

$= \text{Artifact SD fingerprint}$

Bloom filter
- local SD fingerprint
- 256 bytes
- up to 192 features
Advantages & challenges for block-aligned similarity digests (sdbf-dd)

- **Advantages**
  - Parallelizable computation
  - Direct mapping to source data
  - Shorter (1.6% vs 2.6% of source)
  - Faster comparisons (fewer BFs)

- **Challenges**
  - Less reliable for smaller files
  - Sparse data
  - Compatibility with sdbf digests

- **Solution**
  - Increase features for sdbf filters: 128 → 160
  - Use 192 features per BF for sdbf-dd filters
  - Use compatible BF parameters to allow sdbf ↔ sdbf-dd comparisons
sdhash 1.7: sdbf vs. sdbf-dd accuracy

<table>
<thead>
<tr>
<th>Query size</th>
<th>FP rate</th>
<th>TP rate</th>
<th>Query size</th>
<th>FP rate</th>
<th>TP rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>0.1906</td>
<td>1.000</td>
<td>2,000</td>
<td>0.0006</td>
<td>0.997</td>
</tr>
<tr>
<td>1,100</td>
<td>0.0964</td>
<td>1.000</td>
<td>2,200</td>
<td>0.0005</td>
<td>1.000</td>
</tr>
<tr>
<td>1,200</td>
<td>0.0465</td>
<td>1.000</td>
<td>2,400</td>
<td>0.0001</td>
<td>1.000</td>
</tr>
<tr>
<td>1,300</td>
<td>0.0190</td>
<td>1.000</td>
<td>2,600</td>
<td>0.0001</td>
<td>0.997</td>
</tr>
<tr>
<td>1,400</td>
<td>0.0098</td>
<td>1.000</td>
<td>2,800</td>
<td>0.0000</td>
<td>1.000</td>
</tr>
<tr>
<td>1,500</td>
<td>0.0058</td>
<td>1.000</td>
<td>3,000</td>
<td>0.0000</td>
<td>0.999</td>
</tr>
<tr>
<td>1,600</td>
<td>0.0029</td>
<td>0.999</td>
<td>3,200</td>
<td>0.0000</td>
<td>0.998</td>
</tr>
<tr>
<td>1,700</td>
<td>0.0023</td>
<td>0.999</td>
<td>3,400</td>
<td>0.0000</td>
<td>0.998</td>
</tr>
<tr>
<td>1,800</td>
<td>0.0013</td>
<td>0.999</td>
<td>3,600</td>
<td>0.0000</td>
<td>1.000</td>
</tr>
<tr>
<td>1,900</td>
<td>0.0010</td>
<td>0.998</td>
<td>3,800</td>
<td>0.0000</td>
<td>0.998</td>
</tr>
</tbody>
</table>