Digital Investigation 9 (2012) S60–S68

Contents lists available at SciVerse ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Content triage with similarity digests: The M57 case study

Vassil Roussev*, Candice Quates

Department of Computer Science, University of New Orleans, New Orleans, LA 70148, United States

Keywords:
Similarity digest
Forensic triage
Digital forensics
Case study
M57

A B S T R A C T

In this work we illustrate the use of similarity digests for the purposes of forensic triage. We use a case that consists of 1.5 TB of raw data, including disk images, network captures, RAM snapshots, and USB flash media. We demonstrate that by applying similarity digests in a systematic manner, the scope of examination can be narrowed down within a matter of minutes to hours. In contrast, conventional manual examination of all the data may require several days, and its effectiveness relies substantially on the experience of the investigator.

1. Introduction

The volume of data subject to forensic examination keeps growing at an exponential rate as illustrated by data from the RCFL Program Annual Reports.[1] Table 1 shows a cumulative summary of the amount of data being processed by participating RCFLs. The most significant number is in the last column and shows that the amount of data per case has grown at an average annual rate of **28%** from FY 2003 to FY 2010 (the last year for which data is available). Cumulatively, the amount of data per case has grown by 466% in seven years.

To deal with the scalability problem, digital forensic practitioners need to be able to attack the problem from every direction: automate routine examinations, employ more resources (both human and machine), and use more sophisticated processing methods. One of the most promising approaches to reducing the impact of large cases on turn-around time is to perform fast and accurate forensic triage.

We define digital forensic triage as a fast initial screen of the acquired data whose purpose is to identify the most relevant artifacts and help build an overall understanding of the case. In other words, we assume that the evidence has been acquired and the described work is conducted in a lab environment. Thus, our starting point is a collection of evidence containers, such as disk/RAM snapshots and network traces, and (optionally) some pre-built reference databases of known content (files). (In principle, it is feasible to perform similar triage on the spot by bringing in mobile equipment; however, such discussion is outside the scope of this presentation.)

We are interested in a solution that has the following properties:

- Minimizes initial processing; we need answers in minutes/hours as opposed to days/weeks.
- Allows for fast queries; e.g., query TB-sized data in a few minutes.
- Utilizes all available hardware; e.g., it should be possible to concentrate available resources on an urgent case and get a corresponding speedup.
- Generates reliable hints; we do not expect a triage tool to replace more precise methods but whatever results it generates should have a very low false positive rate.

Taken together, these requirements ensure that an examiner can *quickly* screen the content of the data, build an initial understanding of the case, and determine priorities *before* committing resources to deep examination. Speed and reliability are critical to minimizing the cost of

* Corresponding author.
E-mail addresses: vassil@cs.uno.edu (V. Roussev), cequates@uno.edu (C. Quates).

[1] http://www.rcfl.gov/DSP_N_annualReport.cfm.

1742-2876/$ – see front matter © 2012 V. Roussev & C. Quates. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2012.05.012

Table 1
RCFL cumulative case load: FY 2003–2010.

Fiscal year	Processed data (TB)	Number of cases	Avg case (GB)
2003	83	987	84
2004	229	1304	175
2005	457	2977	154
2006	916	3633	252
2007	1288	4634	278
2008	1756	4524	388
2009	2334	6016	388
2010	3086	6564	470

triage. Before we describe our case study, let us briefly consider why current generation integrated forensic environments, such as FTK and EnCase, are fundamentally unsuited for triage work.

The standard approach employed by such deep forensic examination tools is to use a file-centric approach and access data content via the file system API. This is not different from what any other application would do with the notable exception that a forensic tool may attempt to reconstruct files where the metadata is missing (a.k.a. *file carving*). Once a file is retrieved, it is subjected to all available processing (hashing, indexing) and the results are stored in a database for future reference.

This approach is systematic and logical; however, it is also heavyweight and slow. For example, based on the vendor's own numbers,[2] the full complement of FTK's processing shows a long-run throughput of about 10 MB/s on a modern workstation. This is completely inadequate for triage purposes, as it would take days to complete preprocessing. There are several causes for this low processing rate:

- Data carving and reconstruction. Carving is an inexact process that can generate large number of false positives, and often results in significant amount of additional data for processing.
- Indexing. Indexing is a CPU and I/O intensive process. Based on in-house testing, the leading open-source search engine, CLucene, ekes out about 6 MB/s on the hardware we used for our tests (Section 5).
- Use of a transactional database to store the results from processing. It has been argued elsewhere (Roussev, 2011a) that the substantial performance penalty ($\approx$20 times) for using a transaction-oriented store is not justified given the reproducible nature of forensic computing.
- Non-sequential media access. It is well-known that hard drives perform poorly under workloads exhibiting non-sequential access patterns; in our tests, as little as 5% of random reads in the workload can reduce drive throughput to 50% of its maximum. File-centric processing virtually guarantees a non-sequential access pattern, which reduces throughput and increases latency. (SSDs do not suffer from this problem but have capacity and cost constraints that make them expensive for use on large cases.)

[2] http://accessdata.com/distributed-processing.

Thus, it is critical for triage tools to avoid the performance straightjacket of the prevalent deep examination methods and adopt alternative approaches to analyzing data. This means outright rejection of the idea of performing all possible processing and focusing on one or two lightweight methods. It should be clear that the first three performance impediments can be eliminated by simply avoiding the specific processing that causes them (carving, indexing, database transactions).

This leaves the last problem on the list–achieving high throughput media access. This can be approached from one of two complementary directions: a) focus on metadata to reduce the amount of data examined, or b) access data content using only a sequential pattern to minimize latency.

Metadata is the additional information that a computer system must store in order to organize and annotate data content; e.g., for a file system this includes both user-readable data per file, such as name, type, and access time, and system data such as data layout on the physical medium. Data content is generally laid out as a collection of either fixed (disk, RAM), or variable sized chunks (network).

The metadata approach has two major advantages: a) its volume on a typical target is about 1–5% of the overall volume so it can be accessed and processed quickly; b) most metadata is for human use and tends to contain higher-level logical information. The downside of metadata analysis is that it may not be complete, or trustworthy. For example, deallocated objects tend to be reclaimed only when there is demand for the resources but the metadata that allows reconstruction is usually gone. Also, users can manipulate a lot of the metadata directly and could effectively hide their activities (rename files, reset timestamps, edit OS registry).

Scanning raw data content has the advantage of working with the actual data content and largely ignores the metadata. This renders naive hiding techniques ineffective and allows all data present on the medium to be interrogated. The obvious downside is that all data is processed and logical information is not used.

Metadata-based triage has been in use for a long time–to this day, manual file browsing is one of the most common operations by forensic examiners who use their experience to steer the investigation in a promising direction based on observable file system metadata. Timelining and OS registry Carvey (2012) examination are other examples of common metadata-centric triage.

Raw-data triage also has a long history, which probably starts with the use of the strings and grep Unix utilities and block-level crypto hashes. More recently, `bulk_extractor` Garfinkel (2012) has significantly upgraded regular expression search capabilities of raw data while Garfinkel et al. (2010) has demonstrated the use of data sampling to estimate drive content.

The overall goal of this work is to demonstrate the practical use of similarity digests for content-based triage on a non-trivial data set. Our contribution is twofold: a) we use a specific case to quantify the time it takes to answer realistic queries; and b) we show that the differential analysis techniques used in this case are simple and effective, and are readily applicable in the general case.

The rest of the paper is organized as follows: Section 2 provides a brief overview of similarity digests and their basic properties; Section 3 describes the M57 data set; Section 4 introduces the generic content queries used to perform the triage; Section 5 describes the various scenarios, the corresponding triage queries executed, and the observed performance; Section 6 summarizes our discussion.

2. Background: similarity digests and `sdhash`

Since 2006, there has been an increasing interest in the design and use of similarity (or fuzzy) hashes, such as `ssdeep` Kornblum (2006) (based on Tridgell (2012)) and `sdhash` Roussev (2010), derived from Roussev et al. (2006, 2007). Unlike cryptographic hashes, which are designed to test for object identity, similarity schemes seek to uncover objects that have non-trivial similarities in their bit-stream representation.

Specifically, such methods seek to address the following three scenarios Roussev (2012):

- *Identification of embedded/trace evidence.* Given a data object (file, disk block, network packet) we want to be able to detect (traces of) its presence inside a bigger object of arbitrary size, such as a disk image, RAM snapshot, or network capture.
- *Identification of artifact versions.* Given a large set of artifacts (files), identify those that are versions of a set of reference artifacts. The target objects could be executable code (versions of known software), or user-created data objects (documents).
- *Cross-target correlation.* Identify and correlate different representations–on-disk, in-RAM, network flow–of the same/similar object. This would allow automated annotation RAM and network captures without the deep (and costly) parsing and reconstruction that current approaches require.

In Roussev (2011b), a user study has shown that `sdhash` outperforms `ssdeep` by a wide margin in terms of recall and precision rates. Subsequent work Roussev (2012) has lead to a parallelized version, which achieves a significant performance boost. Based on these prior results, we decided to evaluate `sdhash` on an actual case study using the public M57 data corpus.[3]

Detailed conceptual description of the inner workings of `sdhash` is provided by Roussev (2010) and is beyond the scope of this presentation. Instead, we provide a minimal user guide for the tool (version 1.7) and its expected behavior. All `sdhash` code and related material is accessible from the tool's home page: http://sdhash.org/.

2.1. Digest generation

Since version 1.6, building the source code results in two executables: `sdhash` and `sdhash-dd`. The former implements the original `sdhash` algorithm and is to be used to hash files. The latter is a new block-aligned variation described in Roussev (2012) and is suitable for large targets, such as drive images. Note that the output of the two commands would be different for the same target; however, the two versions are complementary and `sdhash` and `sdhash-dd` signatures are designed to be comparable. In a typical scenario in which we wish to search a drive image for a set of reference files, we would apply `sdhash-dd` to the drive image and `sdhash` to the file set and then would compare each of the file digests to the drive digest.

All command line options described here work exactly the same way for both `sdhash` and `sdhash-dd`, so we will not duplicate the explanation.

The general format of the command is `sdhash <target-files>`. Results are base64-encoded and sent to standard output one line at a time (every digest is fitted on exactly one line, with no limit on line length). Example generation of digests for files `sdhash.c` and `sdbf_api.c` yields the following (output trimmed, line breaks added):

```
> sdhash sdhash.c sdbf_api.c
sdbf:02:8:sdhash.c:-
sha1:256:5:7ff:160:1:74:iAEAl
AYAIIkgAAAAB...
sdbf:02:10:sdbf_api.c:-
sha1:256:5:7ff:160:2:74:Da CriTh1trgKmU...
```

The output format is known as *SDBF (Similarity Digest Bloom Filters)* with a recommended file extension of `.sdbf` for the `sdhash` version, and `.sdbf-dd` for the `sdhash-dd` version. Every similarity digest is both self-contained (one line of text) and comparable to any other digest, so the two versions can be freely mixed within the same file.

The output encodes the following pieces of information, separated by colons: magic number (`sdbf`), version (`2`), length of file name (`8`), file name (`sdhash.c`), hash function used to hash features (`sha1`), size of constituent Bloom filters in bytes (`256`), number of subhashes per feature (`5`), bit mask used to derive the subhashes (`0x7ff`), number of features per filter (`160`), number of filters in the digest (`1`), number of features in the last filter (`74`), and base64-encoded sequence of filters (`AEAlAYAIIkgAAAAB`...).

The size of a similarity digest is proportional to the size of the data targets. The in-memory `sdhash` representation is, *on average*, 2.6% of the size of the target (approximately 256 bytes of digest per 9.5 KB of data). After the base64 encoding, it expands to about 3.6% on disk.

By default, the tool runs a single-threaded version of the code; however, the `-p` option can specify a level of thread concurrency desired. To run the previous example with two threads we would use:

```
> sdhash -p 2 sdhash.c sdbf_api.c
```

For maximum performance, the parallelization factor should match the number of hardware-supported threads on the platform (we used `-p 24` in all of our tests in Section 5). For the `sdhash` version, the execution is file-parallel, which means that each file is hashed sequentially but multiple files can be hashed concurrently. The `sdhash-dd` computation is block-parallel–it splits each target into 16KiB blocks, hashes them in parallel, and concatenates the results.

From a user's perspective, `sdhash-dd` generation works the same way but produces slightly different looking digests. For example:

```
> sdhash-dd sdhash.c sdbf_api.c
```

[3] http://digitalcorpora.org/corpora/scenarios/m57-patents-scenario.

```
   sdbf-dd:02:8:sdhash.c:-
sha1:256:5:7ff:192:1:16384: 4A:iAEAlAY...
   sdbf-dd:02:10:sdbf_api.c:-
sha1:256:5:7ff:192:1: 16384:C0:DaSr...
```

The main difference in the digest is the inclusion of a block size (`16384`) and the number of features for every filter (`0x4A`). The in-memory `sdhash-dd` representation is 1.6% of the size of the target (258 bytes per 16,384 bytes of data).

2.2. Digest comparison

Before we describe how to perform digest comparisons, we should note that there is no difference in the way `sdhash` and `sdhash-dd` perform the comparison–they run the exact same code.

The result of digest comparison is an integer between **−1** and **100**. Despite the range, a positive result is *not* interpreted as a percentage of commonality but as a confidence measure. Thus, a result of *zero* indicates that the two objects are judged to be uncorrelated, whereas 100 means the tool is certain that the two objects are correlated; *not* necessarily identical.

Put another way, comparing two identical objects would produce a score of 100; however, a score of 100 is not a definitive indication that the objects are identical. This is by design so that the score could be used both for containment and resemblance queries. For example, if we copied several KB from the beginning of a file and compared it to the whole, the result would be 100 although the two objects are not identical. If identity testing is required, then crypto hashes should be employed as a separate step.

The *do-not-know* value of **−1** indicates that at least one of the digests does not contain enough features to make a trustworthy comparison. This usually happens with small (<4 KB) and/or sparse files.

The main comparison option is `-c` and can be invoked with one or two arguments. The single argument format (`sdhash -c <sdbf-file>`) instructs `sdhash` to load the digests contained in the given file and compare all possible pairs. For examples, if the `sdbf` file contains five files, the tool will perform 10 comparisons since *sdhash*(a, b) = *sdhash*(b, a). Here is a simple sequence of shell commands that would compare two files:

```
> sdhash sdhash.c sdbf_api.c
> test.sdbf > sdhash -t 0 -c test.sdbf
sdhash.c|sdbf_api.c|000
```

The output consists of three columns which contain the names of the two files and the result from the comparison; the shown field separator is customizable. Note that we used the `-t option`, which specifies the minimum threshold for output. By default, it is set to one, so only pairs with positive scores will be reported. The -p options works as before to specify parallel execution.

The two argument comparison format `sdhash -c <sdbf-file1> <sdbf-file2>` instructs the tool to compare every hash from the first file to every hash from the second one. Thus, if the first file has 10 hashes and the second one has 20, a total of 200 comparisons would be performed. In the output, every hash name is prefaced by the name of the `sdbf` file:

```
   > sdhash sdhash.c sdbf_api.c > test1.sdbf
   > sdhash sdhash.c sdbf_api.c > test2.sdbf
   > sdhash -c -t 0 test1.sdbf test2.sdbf

test1.sdbf:sdhash.c|test2.sdbf:sdhash.c|100

test1.sdbf:sdhash.c|test2.sdbf:sdbf_api.c|000

test1.sdbf:sdbf_api.c|test2.sdbf:sdhash.c|000
   test1.sdbf:sdbf_api.c|test2.sdbf:sdbf_api.
c|100
```

Finally, one option that we frequently used to speed up our queries in the case study is `-s`. It allows us to run comparison operation by using only a sample of the available digest. Assuming that `ref-files.sdbf` is a reference set of `sdbf` file hashes, and `disk.sdbf-dd` is the digest for a disk drive image, we would use sampling in the following fashion:

```
> sdhash -c -s 4 ref-files.sdbf disk.sdbf-dd
```

In this case, each digest from `ref-files.sdbf` is loaded and reduced to no more than four constituent filters, which we expect to represent about 40 KB of the original file. The sample is then compared to the full `disk.sdbf-dd` digest (no sampling) and the result is calculated as it is normally done. This technique is very effective if we need a quick test for the presence of *whole* known files in larger containers–disk/RAM images and network traces.

2.3. Interpretation

Based on the discussion in Roussev (2010, 2011b) and our experience in this work, the results from the `sdhash` comparisons are best interpreted along the following scale:

- *Strong (range: 21–100).* These are reliable results with very few false positives. When used to evaluate resemblance of two comparable in size objects (files) the number is *loosely* related to the level of commonality but this is not a guarantee. When used as part of a containment query (find a small object inside a bigger one), the number can vary widely depending on the particular position of the embedding. In other words the larger object may contain the small one 100% but the score may be as small as 25. Roussev (2011b) has a controlled study that quantifies this behavior.
- *Marginal (11–20).* The significance of resemblance comparisons in this range depends substantially on the underlying data. For many composite file types (PDF, MS Office) there tends to be some embedded commonality, which is a function of commonly used applications leaving their imprint on the file; we have observed this on occasion even with JPEG files that contain lots of (Adobe Photoshop) metadata. In that sense, the tool is not wrong but the discovered correlation is usually not of interest. Other embedded artifacts, such as fonts, are also among the discovered commonalities but are rarely significant. For simpler file types, results in this range are much more likely to be significant and should be examined in decreasing order until the false positives start to dominate.

- *Weak (1–10).* These are generally weak results and, typically, most would be false positives. However, when applied to simple file types, such as text, scores as low as five could be significant (Roussev, 2011b).
- *Negative (0).* The correlation between the targets is comparable to that of two blobs of random data. Special care needs to be taken when comparing large targets to each other as discovered commonality could be averaged out to zero. For example, if two 100 GB have 1 GB in common, the tool will discover that fact but when averaged with the results from the remaining 99 GB, the final score will almost certainly be zero, and definitely not more than one.
- *Unknown (-1).* This is a rare occurrence for files above 4 KB unless they contain large regions of low-entropy data. The absolute minimum file size that `sdhash` will consider hashing is 512 bytes. If a case requires the comparison of lots of tiny files, `sdhash` is likely the wrong tool.

When comparisons involve block digests, the level of false positives is somewhat elevated so the threshold of significance tends to be a little higher. For example, in our study (`sdhash vs. sdhash-dd digests`) we completely ignored results in the 1–10 range.

Finally, comparing the two versions of an object's digest yields a strong result but it is almost never 100. In other words, if $s_1 = sdhash(x)$ and $s_2 = sdhashdd(x)$, then $sdhash(s_1, s_2) < 100$ in the general case. To illustrate, consider two files–`1M-01.rnd` and `1M-02.rnd`–that have nothing in common. Each contains 1 MB of independently generated pseudo-random data. Then, the following is a typical result:

```
> sdhash 1M-01.rnd 1M-02.rnd > 1M.sdbf
> sdhash-dd 1M-01.rnd 1M-02.rnd > 1M.sdbf-dd
> sdhash -c -t 0 1M.sdbf 1M.sdbf-dd
1M.sdbf:1M-01.rnd|1M.sdbf-dd:1M-01.rnd|049
1M.sdbf:1M-01.rnd|1M.sdbf-dd:1M-02.rnd|000
1M.sdbf:1M-02.rnd|1M.sdbf-dd:1M-01.rnd|000
1M.sdbf:1M-02.rnd|1M.sdbf-dd:1M-02.rnd|050
```

Since in the world of similarity we work with ranges, the above behavior is not a problem as long as the `sdhash` vs. `sdhash-dd` comparisons yield strong correlations. They do.

3. The M57 data set

The classic problem in discussing digital forensic cases is the fact that actual cases have obvious privacy constraints, whereas most publicly available data sets are very limited in scope. The only exception to the latter is the M57 Patents scenario created by the Naval Postgraduate School. It features the fictitious *m57.biz* patents research company, whose employees' actions are scripted, performed, and recorded on a private network. Thus, the data set constitutes a realistic exercise yet it carries no restrictions and provides the best available public platform for peer-reviewed research.

The 2009-M57-Patents scenario encompasses a 17-day period between November 16th, 2009 and December 11, 2009 (excluding weekends and holidays). The company has four employees–Pat (CEO), Terry (IT administrator), Jo (patent researcher) and Charlie (patent researcher)–who are engaged in a variety of legal and illegal activities. For the purposes of the scenario, employees interact with several other personas created outside of the m57 company to mimic real world interactions. These represent friends, acquaintances, clients, and other individuals in contact with the m57 employees.

The overall amount of data is 1.5 TB in raw form (460 GB compressed) and consists of the following data.

3.1. Disk images

Each persona's workstation hard drive was imaged daily,[4] except for weekends and holidays, using the aimage tool provided in the `AFF library`http://afflib.org. Additionally, at the end of the scenario the hard drives are imaged again. All of the hard disk images use the NTFS file system. There are **78** disk images in the set varying in size between 10 and 40 GB (raw) for a total of **1,423 GB** of data.

3.2. RAM snapshots

Similar to the disk images, the RAM contents of each workstation are captured daily,[5] except for weekends and holidays. All four machines run different versions of MS Windows with the individual machines having the following amounts of RAM–Charlie: 1 GB (XP), Terry: 2 GB (Vista), Jo: 512 MB (XP), Pat: 256 MB (XP). There are **84** RAM snapshots in the set with a total size of **107 GB**.

3.3. Network traffic

Network data consist of **49** packet captures[6] using the gateway's interface for a total of **4.6 GB** of raw data. Data covers every day the scenario was in operation, including weekends and any holidays that occurred during the scenario.

3.4. Device images

Images of the four USB devices used during the period are imaged at the end of the scenario (4.1 GB total).

3.5. Kitty material

There is an additional reference set of known "kitty porn" material, which simulates contraband. The set consists of 125 JPG images (43 of which are lower resolution versions of the originals) and six movies. Total amount of the set: 224 MB.

4. Content-centric triage

Recall that similarity digests are designed to solve two content correlation problems–resemblance and

[4] https://domex.nps.edu/corp/scenarios/2009-m57/drives/.
[5] https://domex.nps.edu/corp/scenarios/2009-m57/ram/.
[6] https://domex.nps.edu/corp/scenarios/2009-m57/net/.

containment. That is, we can use `sdhash` to determine if two objects of comparable size resemble each other, or if (pieces of) a smaller object are contained inside a bigger one. In the former scenario, we would most commonly compare one file to another file, or relatively small volume images, such as portable flash drives. While there is no explicit limitation that prevents us from comparing, for example, two hard drives the information gleaned would be rather minimal (e.g., two MS Windows drives are similar) and not worth the relatively long computation. In the latter scenario, we would search for relatively small artifacts–blocks, files, network packets–inside bigger targets, such as disk images. We should emphasize that the distinction is conceptual and is helpful in formulating useful queries and correctly interpreting the results; however, the tool is invoked the same way and works the same way in both cases.

For these triage cases, we use four types of queries we consider representative of the style of inquiry in the general case:

- *File vs. HDD.* The purpose of this query is to establish whether (pieces of) a particular file can be found on a drive image. The drive would typically be hashed with `sdhash-dd`. The query will cover the entire image and can find deleted, or partially overwritten files. If the file is present in its entirety, the sampling feature of `sdhash` can be used, which allows us to query for a small part of the file, thereby speeding up the search process. If sampled queries do not yield results, we can fall back to a complete query.
- *File vs. RAM.* The purpose of this query is to establish whether traces of a particular file can be found in a RAM snapshot. The RAM image would typically be hashed with `sdhash-dd`, using the page size as the block size (4KiB). Typically, we would search for all executables from a HDD image to establish what programs have run. Alternatively, we can search for data files as they tend to persist in the file cache until space is needed.
- *File vs. pcap.* The purpose of this query is to establish whether traces of a particular file can be found in a network capture. This is somewhat less reliable as a lot of network traffic gets automatically transformed (base64, zip) and would require pre-processing to gain the maximum information. Nevertheless, our case study shows that, in seconds, we can still identify useful data even without any pre-processing, or flow reconstruction.

5. The M57 scenarios

The main goals of our triage process are to quickly build an overall picture of the case, narrow down the focus of the inquiry, and provide strong hints as to what the final outcome of the inquiry might be. We fully expect that precision methods involving deep inspection will be a follow-up step to unambiguously establish the relevant facts. In this case study, we have the unique benefit of knowing the ground truth by having the complete sequence of user actions.

The overall process is very simple–we generate the similarity digests for all targets and all queries (files of interest) and then systematically apply the queries to the targets. Since this is a triage process, we place a particular emphasis on the speed with which we can get actionable results, as well as their accuracy.

All tests were run on one of two similarly configured systems. The slightly faster system is a server with two six-core Intel Xeon CPUs @2.93 GHz with hyper-threading, for a total of 24 hardware threads. The host is equipped with 72GiB of RAM @800 MHz and has hardware-supported RAID 10 with benchmarked sequential throughput of 260 MB/s. All presented numbers are stable averages over multiple runs (unless otherwise noted).

5.1. *Pre-processing:* `sdhash` *generation*

The first step is to generate the similarity digests of the targets. For all disk images, we use the block-aligned version of the digests (`sdhash-dd`), which splits the target into 16KiB blocks, generates a digest for each block, and concatenates them. The main advantage of this approach (versus mounting the image and hashing individual files) is speed: block hashing is parallelized and disk access remains sequential. In Roussev (2012) it is shown that a 24-threaded machine similar to the one we use can achieve throughput of 370 MB/s on cached data, which makes the hashing process I/O-bound on most systems. The other advantage is that we do not depend on our ability to process the file system layout and do not need to perform carving to retrieve deleted data.

For the RAM images, we used the sdhash-dd version of the algorithm, except we set the block size to 4KiB. The rationale here is obvious–the virtual memory system uses 4KiB as the default page size.

For the USB drives we generated `sdhash-dd` digests for each image. For specific queries, we also hashed all files extracted from the mounted image.

The kitty material files were hashed with the base sdhash algorithm, which is suitable for individual files.

Table 2 provides a summary of the observed digest generation performance for each of the sets–hard disk images, RAM snapshots, network captures, USB drives, and reference file set–providing the cumulative size, the execution time, and the throughput rate. The main point here is that the performance is I/O-bound, so the digest generation can proceed in parallel with the acquisition process. In practice, this can be achieved by utilizing an imaging tool like `dcfldd` (http://dcfldd.sf.net/), which can provide two parallel streams–one for creating a forensic image and one

Table 2
Summary: sdhash generation performance.

Data Set	Size (GB)	Time (min)	Rate (MB/s)
HDD	1423.0	168.00	143
RAM	107.0	10.70	166
Network	4.6	0.40	196
USB drives	4.1	0.45	155
Kitty	0.2	0.08	45
Total	**1538.9**	**179.63**	**143**

for the hash generation. Thus, with proper planning and sufficient hardware, we can get the similarity digests of the target at no extra latency and can begin querying them immediately after the end of the cloning process.

In comparison, current state-of-the-practice forensic tools would be just starting their pre-processing. If we assume (optimistically) that each of the 78 HDDs could be processed (on average) in 10 min and each of the 84 RAM snapshots in 5 min, we end up with a total estimate of about 20 h. That is, we have to wait that long before we can start work–clearly, this does not work in the context of triage.

5.2. *Case #1: contraband*

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". The suspicion arose when Jo's old computer was replaced on Nov 20 and subsequently sold online. We also know that Jo might have tried to cover his tracks when police seize the computers on the last day of the scenario.

The following are several relevant questions that can help us get an overall picture of the case:

- Were any M57 computers used in contraband?
- If so, when did the incident happen?
- Is there evidence of intent?
- How was the content distributed?
- Was any of the content sent outside the company network?

To answer the above questions we start by querying all of Jo's disk images with the kitty set. Since the full `sdhash` comparison compares every digest component of the query files to every digest component of the target, it would take 55 min to run through 21 disk images (260 GB of raw data). However, we could do a much faster scan by *sampling* the query digests, effectively looking only for pieces of the original files.

For example, if we use a sample of four consecutive filters (digest components) we would be searching for traces of about 40 KB, on average, of the query file in the target. Such a sampling query (using the **-s** option of `sdhash`) needs only 123 s to complete. We can be even more aggressive and use a sample size of two, which in this case yields the same results, at the cost of only 96 s for all of Jo's hard disk images.

Fig. 1 provides a histogram of the number of file matches for each of the dates and provides an excellent basis for forming an initial hypothesis (`sdhash` is run with a threshold score of 10). It appears that 124 files of the kitty set were copied onto Jo's computer on 11/18, with five more added on 11/20. When Jo's computer is replaced, all kitty material disappears until 11/24 when 130 files are again placed onto the new machine. Starting on 12/03, the number of files starts dropping gradually, which could be explained by the files being deleted by the user and then being gradually overwritten by the system. (In reality, Jo copies the files into a TrueCrypt volume and deletes them.) On the last day, 12/11, we see that kitty files reappear, which turns out to be function of the user trying to cover up his tracks. Querying all the 78 disk images for traces of contraband (*including* Jo's) using a sample size of two takes 7 m 58 s, whereas the same query with a sample of four takes 11 m 10 s. (The complete query for all disks would take 5 h 10 m and would be more appropriate as part of

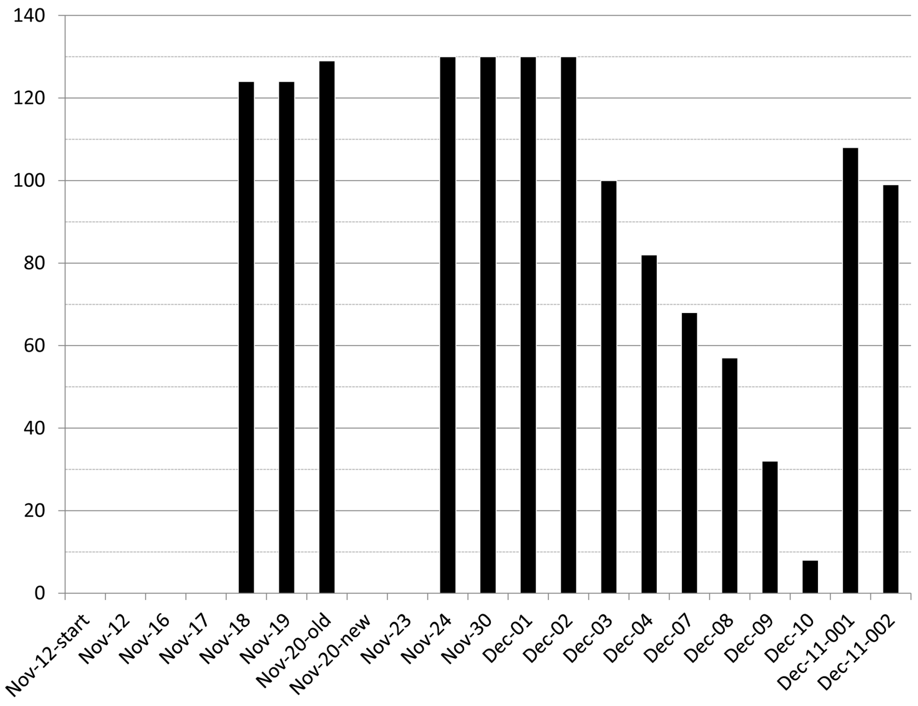

Fig. 1. Frequency of contraband files on Jo's computer.

a deeper investigation.) The results do not reveal any new matches on any of the other machines. Similarly, a query of the network traces (1 m 58 s) does not show any hits. From the scenario script we know that, indeed, the kitty files were never placed on any of the other machines and were never transferred over the network. A review of Jo's USB drive finds the source kitty images in seconds.

In summary, in approximately 20 min, we were able to reach the following *preliminary* conclusions: a) yes, the initial suspicion of contraband is correct; b) only Jo's computer is involved; c) the images were likely introduced on Nov 18; d) the USB drive is the apparent mode of transmission; and e) there is every reason to believe that somebody (most likely Jo) has deliberately placed kitty material on his computer.

We also performed an examination of Jo's RAM images in the following fashion: for each day we mounted the HDD image, hashed all executable files and used them to query the RAM snapshot of the same day (18 m 06 s). We found strong indication of Jo downloading, installing and using TrueCrypt. The timeline below shows the relevant matches and the corresponding score.

```
12/03
.../Downloads/TrueCrypt Setup 6.3a.exe 092
.../TrueCrypt Format.exe 090
.../TrueCrypt Setup.exe 092
.../TrueCrypt.exe 092
12/04
.../Downloads/TrueCrypt Setup 6.3a.exe 063
.../TrueCrypt Setup.exe 063
12/09
.../Downloads/TrueCrypt Setup 6.3a.exe 084
.../TrueCrypt Format.exe 079
.../TrueCrypt Setup.exe 084
.../TrueCrypt.exe 090
12/10
.../TrueCrypt.exe 092
12/11 - pre-raid
.../TrueCrypt Format.exe 086
.../TrueCrypt.exe 079
```

All of these match the actual sequence of events as described in the M57 script. In summary, we were able examine and correlate all the 21 disk and 18 RAM images to piece together the essential outline of the case, and we used almost no case-specific knowledge. Total triage time: *40 min*.

5.3. Case #2: eavesdropping

In the eavesdropping scenario, it is suspected that somebody is spying on the CEO (Pat) electronically. Our starting hypothesis is that a rogue process was introduced on his computer. Since we lack more specific information with respect to timing, we will examine all of Pat's RAM snapshots, and attempt to establish which executables are in memory.

For this case we generated some additional file hashes–we mounted each of Pat's disk images and hashed all individual files; this took 1 h 30 m for 18 images (260 GB).

For each day, we compared the memory images from Pat's computer against the `sdbf` hashes of the .exe and .dll files on the disk images from his computer to establish what code was run on the machine. To obtain a baseline, we also compared all the memory images against the executable files from the starting disk image, which we know is clean. The resulting difference gives us the set of executables that are not present in the base image but were run on Pat's computer.

Using a sampling size of four filters, it took 8 m 49 s to compare the base image executables to all 21 RAM snapshots, and 10 m 30 s to compare the executables from the same-day disk image to the corresponding RAM snapshots. (For reference, running the complete version of the latter with no sampling was measured at 2 h.)

Based on the difference results and a few lines of light scripting we were able to produce the following timeline of potentially relevant processes, as well as some routine activities that can be easily discerned. For each day, we also give the size of the difference (number of files to be examined), which naturally grows over time. For brevity, weekends and dates with no interesting results are skipped.

```
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.
Other programs from 11/16 still present.
11/20, [289]
Looks like Windows Update has run with many new dlls in the _restore and SoftwareDistribution folders.
11/23, [561]
Windows Update was run (again)
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 is present:
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.
```

At this point it certainly appears that a keylogger was installed on Pat's machine on 12/03 and was removed on 12/07. If Pat did not install RealVNC on 12/07, it could also have been used to look over his shoulder.

Total triage time: *30 min* for queries (plus 1 h 30 m for additional file hashing).

5.4. *Case #3: corporate espionage and extortion*

There is suspicion that somebody has leaked company secrets. We turn our attention to Charlie's computer and compare the executables from the disk image with the corresponding RAM snapshot of the day. Using a sample size of four, we complete the comparison of all 18 RAM images in 31 m 02 s.

We identified the presence of the "Cygnus FREE EDITION" hex editor on 11/24, 11/30, 12/02, 12/03, and 12/10; further, we found a program called "Invisible Secrets 2.1" on 11/19, 11/20, 11/24, 11/30, and 12/02. It has dlls named "blowfish", "jpgcarrier", and "bmpcarrier," which supports the suspicion that it is a steganography tool. On Charlie's USB drive we find a program "insecr2.exe," which we can find in the network trace for 11/19 suggesting it was downloaded on that day.

Our next step is to identify what data might have been exfiltrated. We searched the USB drive for documents (everything but executables) and find the following interesting collection of files:

```
    /microscope.jpg
    /microscope1.jpg
    /astronaut.jpg
    /astronaut1.jpg
    /Email/Charlie_..._Sent_astronaut1.jpg
    /Email/other/
Charlie_..._Sent_microscope1.jpg
```

Upon examination, the "microscope" and "astronaut" pairs of images appear identical, yet their hashes are different suggesting that they might have been used as carriers for stego images. Examination of accompanying emails confirms the hypothesis.

In looking for other possible exfiltration carriers, we find that 7-zip is present and was installed on 11/24. On the USB drive we find a password-protected archive `01.zip`, which apparently contains two files–*us005026637-001.tif* and *us006982168-001.tif*. By searching for all files from the USB drive in all Charlie RAM images (54 s with sample size of four) we find the two files in memory on 11/24, 11/30, and 12/02. Correlating the zip file with mail traffic reveals that it is part of an extortion scheme, rather than exfiltration.

Total triage time: *40 min*.

6. Conclusions

In this work, we used a sizeable case study to demonstrate the utility of similarity digests as a triage tool. In particular, we showed that a single and simple data correlation tool– `sdhash`–can provide a systematic and efficient path to triage with minimal assumptions and knowledge of the case. (Indeed, we omitted additional specific information available in the form of emails, which make the cases substantially easier to solve.) Our experience supports the following conclusions:

- Similarity digests offer broad data correlation capabilities that allow an investigator to reliably link data and code artifacts across different types of evidence sources, such as disk images, RAM snapshots, and network traces.
- The current `sdhash` implementation offers digest generation rates that allow the hashing to be performed in parallel with disk target acquisition; this enables immediate triage queries on the target. We expect this approach to be pushed further by enabling queries for known content to be performed *while* the acquisition is running.
- We have found the correlation queries offered by `sdhash` very effective and time efficient approach for building an initial framework of understanding for the cases studied. While this does not eliminate the need for follow-up deep analysis to confirm the preliminary conclusions, it can give a significant head start to the investigation by pointing it in the right direction.
- The overall approach of using digests is very simple and scalable, which allows for it to be applied in a systematic and automated fashion to all cases; further, the results are intuitive and easy to interpret, which implies that it can be quickly adopted into practice.

We emphasize that while the style of content-based triage supported by `sdhash` is unique, it is also complementary to other triage techniques, especially metadata ones. We believe that we are seeing the beginning of a new approach toward large-scale investigations, which relies heavily on a combination of fast, lightweight triage methods applied aggressively from the moment evidence acquisition starts. In the fullness of time, we expect deep analysis, as currently practiced, to be applied ever more selectively as its performance costs are becoming unbearable.

References

Carvey H. Windows forensic analysis toolkit. 3rd ed.;, ISBN 978-1597497275; 2012.

Garfinkel S, Nelson A, White D, Roussev V. Using purpose-built functions and block hashes to enable small block and sub-file forensics. In: Proceedings of the tenth annual DFRWS conference (DFRWS'10). Portland, OR; Aug 2010.

Garfinkel S. Digital media triage with bulk data analysis and bulk extractor. Working paper, http://simson.net/ref/2011/bulk_extractor.pdf [accessed: 20.02.12].

Kornblum J. Identifying almost identical files using context triggered piecewise hashing. In: Proceedings of the 6th annual DFRWS. Lafayette, IN; Aug 2006.

Roussev V, Chen Y, Bourg T, Richard GG III. md5bloom: Forensic filesystem hashing revisited. In: Proceedings of the 6th annual DFRWS conference (DFRWS'06). West Lafayette, IN; Aug 2006.

Roussev V, Richard GG III, Marziale L. Multi-resolution similarity hashing. In: Proceedings of the 7th annual DFRWS conference (DFRWS'07), Pittsburgh, PA; Aug 2007, doi:10.1016/j.diin.2007.06.011.

Roussev V. Data fingerprinting with similarity digests. In: Chow K, Shenoi S, editors. Research advances in digital forensics, vol. VI. Springer, ISBN 978-3-642-15505-5; 2010. p. 207–26.

Roussev V. Building open and scalable digital forensic tools. In: Sixth international workshop on Systematic Approaches to Digital Forensic Engineering (IEEE/SADFE 11), May 2011a, Oakland, CA.

Roussev V. An evaluation of forensics similarity hashes. In: Proceedings of the eleventh annual DFRWS Conference. New Orleans, LA; Aug 2011b. p. 34–41.

Roussev V. Scalable data correlation. In: Eighth annual IFIP WG 11.9 international conference on digital forensics. http://roussev.net/sdhash/parallel-sdhash-IFIP-12.pdf [accessed 20.02.12].

Tridgell A. Spamsum README, http://samba.org/ftp/unpacked/junkcode/spamsum/README [accessed 20.02.12].