hashdb Toolset

The hashdb toolset consists of the following:

hashdb_manager tool
- Import hash database from dfxml input.
- Copy, append, merge hash databases.
- Track database attribution.
- Provide remote lookup service over TCP.

hashdb library
- API compiles and links with user code via header file hashdb.h and library libhashdb.
- The bulk_extractor hashid scanner interfaces with this library.

Availability

hashdb Beta is available on GitHub at https://github.com/simsong/hashdb.

hashdb wiki is available at https://github.com/simsong/hashdb/wiki.

Papers

Distinct Sector Hashes for Target File Detection

Using Purpose-built Functions and Block Hashes to Enable Small Block and Sub-file Forensics

hashdb_manager

The hashdb_manager tool provides commands for creating and managing hash databases and for providing a hash database query service over a socket:

Copy	Import a DFXML file into a hashdb or copy hashes from one hashdb into another.
Remove	Remove hashes in a DFXML file or a hashdb from another hashdb.
Merge	Merge the contents of two hashdbs into a new third hashdb.
Rebuild Bloom	Rebuild the bloom filter of a hashdb to a different size in order to improve lookup performance.
Export	Export the hashes in a hashdb out to a DFXML file.
Info	Print out information about a hashdb.
Server	Provide a query lookup service for a hashdb.

In hashdb v1.0, query lookup services are provided by a separate hashdb_checker tool. This tool provides the following services:

query_hash	Query a hashdb for hash values that are in a hashdb.
query_source	Query for hash source information for a list of hash values.

hashdb

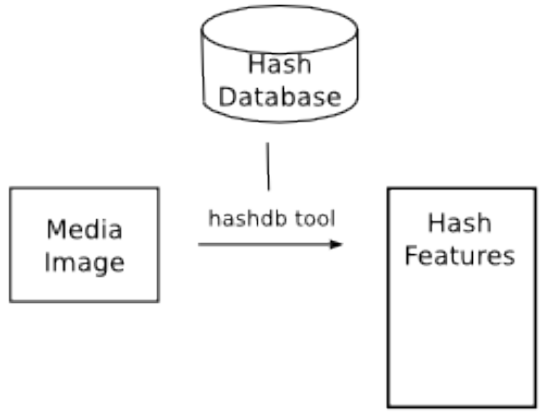
hashdb is used to find blacklist data in raw media by using chunk hashes.

hashdb provides tools for managing databases of chunk hashes and for performing hash lookups. Chunk hashes are hashes computed from chunks of data and are typically 4K in size.

hashdb provides two roles:
- Creating and maintaining a hash database
- Scanning media for matching hash values

1: Create a hash database of chunk hash values from blacklist files:

2: Scan for blacklisted hash values in media:



Creating a hashdb

The hash database is populated with blacklist chunk chunk hash values from named sources. hashdb imports chunk hash values formatted in DFXML. The md5deep file hashing tool generates chunk hash values from files in DFXML. Creating a hash database takes two steps:

- Create a DFXML file of chunk hashes
- Import chunk hashes into a hash database

This example uses the md5deep tool to create DFXML file "my_dfxml_file" containing chunk hashes from files under directory "my_dir" and then imports those hash values into a new hash database named "my_hashdb":

```
md5deep -p 4096 -d -r my_dir > my_dfxml_file
hashdb_manager --copy my_dfxml_file my_hashdb
```

Scanning for Hashes using bulk_extractor

bulk_extractor includes a scanner for finding matching chunk hash values. Matching hash values are stored in the "identified_blocks.txt" feature file.

This example scans image file "my_imagefile" for chunk hash values that match values present in the "my_hashdb" hash database and places matches in the "outpath/identified_blocks.txt" feature file:

```
bulk_extractor -S query_type=use_path \
-S path=my_hashdb -o outpath my_imagefile
```

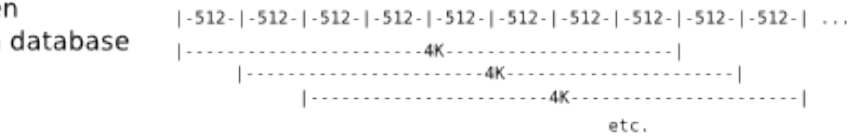
Terminology

Chunk Hash

A chunk hash is a hash of a chunk of data, typically of 4K of data. We copy chunk hashes into a hash database and then search for them in media using the bulk_extractor scanner.

To increase the probability of finding chunk hashes in sector-based images, the bulk_extractor hashid scanner generates chunk hashes at each sector boundary.

The following illustrates how the hashid scanner would create chunk hashes from 4K chunks of data aligned on 512 byte sector boundaries:



Hash Source

Hashes that populate a hash database may be imported from many files and from many repositories. The following is used to track where imported hash values are sourced from:

- Repository name
- Filename
- Offset in the file

A hash value can have multiple sources. It could appear in another repository, in another file, or fin muliple places in a file.

When the bulk_extractor hashid scanner finds matching hashes, it only records the matching hash value. To determine the source of the hash values, a post-processing source lookup is performed. In hashdb v1.0, this lookup is performed using a separate hashdb_checker tool. The following example illustrates looking up hash sources from hash database "my_hashdb" from feature file "identified_blocks.txt":

```
hashdb_checker --query_source -q use_path \
-p my_hashdb identified_blocks.txt
```

Database Attribution

Attribution is provided through history logs. A command log is entered for each command that modifies a hashdb. When a hashdb is copied or when two hashdbs are merged, the old command logs are retained.

Bloom Filters

Hash databases can be large. Bloom filters may be used to improve performance during hash lookups. Bloom filters quickly indicate when a hash is not in the hash database, which is expected to be the case most of the time. If the bloom filter indicates that a hash might be in the database, then a database lookup is required for verification.

If a Bloom filter is too small, it will generate too many false positives to be helpful. If a Bloom filter is too large, it will unnecessarily consume memory.

Tune the size of your Bloom filter based on the expected number of hash values that will be in you hash database.

Usage

Usage text and examples are available by typing the following:

```
hashdb_manager -H
```