FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Sub-file hashing Windows OS

Creating sub-file hash database for forensic analysis

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software and Information Engineering

by

David Tichy
Registration Number 00953189

to the Faculty of Informatics

at the TU Wien

Advisor: Dipl.-Ing. Dr. Martin Schmiedecker

Vienna, 3rd June, 2018

David Tichy

Martin Schmiedecker

Erklärung zur Verfassung der Arbeit

David Tichy
Lassallestrasse 40/2/14, 1020 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. Juni 2018

David Tichy

Acknowledgements

I would like to thank Dr. Martin Schmiedecker and Dr. Katharina Krombholz who filled me with enthusiasm for the topic digital forensics and especially in context of this work Dr. Martin Schmiedecker who helped me to choose this topic.

Also I'd like to thank all of my study colleagues, who supported me during the years of the bachelors program at the Technical University of Vienna.

Abstract

For efficient forensic analysis of a given hard drive, it is important to reduce the quantity of data, which should be processed. This can be achieved by white listening known files and excluding them from further analysis. The National Institute of Standards and Technology (NIST) provides a database of known files and corresponding MD5 and SHA1 hashes, called the National Software Reference Library (NSRL), which is already in use by distributed forensic software like EnCase.
Based on the paper PeekaTorrent [SN16], by Sebastian Neuner, I used the technique of *hash-bases carving*, which is executed by the tool *hashdb* mentioned in the paper, to create sub-file hashes and extend the NSRL for known files in operational systems.

Contents

CHAPTER 1

Background

Digital forensics involves the extraction and analysis of digital media to identify potential evidence. Over time researchers came up multiple disciplines to extract data like file system extraction, to gather data and information from hard drives (file system forensic) [Car05], memory forensics, for extracting RAM data, or network forensics. Additionally, metadata can also be extracted, for example when a certain file has to be created, which later on would lead to further increase of the volume of data a forensic investigator would've to analyse.

Another issue is the increasing storage size of future hardware (Hard drives, RAM), and the fact, that home hardware isn't the only source of forensic analysis. Smartphones, notebooks or Internet of Things (IoT)-devices are reliable sources for forensic nvestigations. All those facts combined will make manual forensic analysis in future more difficult, so in recent years some approaches were developed to oppose these challenges. One approach is to reduce the volume of data, which should be manually processed by marking notable files (blacklisting files), or exclude known (benign) files (whitelisting files) from analysis. A way to realise this approach is to collect common files, create hashsums of these and prepare a list including the hashvalues of hem. The NSRL is such a list with known files.

To exclude files from the forensic a nalysis in this approach, the whole file mustn't be changed. I case a file is partially manipulated or incomplete, the file will not be excluded from manual analysis. With *hash-bases carving* we can encounter this issue, by hashing data blocks of a given file instead of hashing the complete one. [SLGa14] The open-source tool *hashdb* provides the algorithms for *hash-bases carving* and a custom database to perform efficient hash lookups. For example a 2 TB hard drive, New Technology File System (NTFS) formattet with 4KB blocksize, contains over 488 million sectors, which has to be processed fast for further forensic analysis.

For this project I combined the NSRL with the *hash-bases carving* technique to reduce manual forensic analysis of a given image by extracting known operational system files and inserting the sub-file hashes of those operational system files into a *hashdb* database. The generated databases can be used afterwards in the forensic tool *bulk_extractor*.

- Which operational system should be used for efficent support of forensic work
- Are there enough files, which are found in the NSRL dataset and can they be used in data processing for forensic analysis
- Can the process, of generating sub-file hashes, been easily reproduced or even automated

CHAPTER 2

Preanalysis

As targeted operational systems I decided to take Windows 7, Windows 10 and Windows Server 2016. In the context of my scientific work for SBA Research, I evaluated, that Windows is the most used desktop operational system and therefore frequently seen in forensic work.
To carry out the project I decided to work on an UNIX operational system, Arch Linux exactly, for three reasons. I needed the open-source tool *hfind*, which is only available for UNIX operational systems, the open-source tool *hashdb* (https://github.com/NPS-DEEP/hashdb) works flawless without further configuration and the out-of-the-box provided common UNIX tools for shell support the implementation of the project. The NSRL datasets, Version 2.57 [1], were seperated into smaller sets, but for this project I needed the modern dataset [2], which contains data of applications from the year 2000 to present. With a size of 3.2 GB this is also the largest dataset.

[1] https://www.nist.gov/itl/ssd/cs/current-rds-hash-sets
[2] https://s3.amazonaws.com/rds.nsrl.nist.gov/RDS/current/RDS_modern.iso

CHAPTER 3

Implementation and methods

For this work I performed following steps:

1. Obtain a installation of Windows 7, Windows 10 and Windows Server 2016
2. Install every Windows version on a seperate virtual disk
3. Extract a raw copy of each virtual disk and save them in seperate RAW files
4. Obtain the NSRL dataset
5. Mount the RAW file, which will be worked with, read only
6. Identify all files from the image, which are found in the glsnsrl dataset and generate a list with the found files
7. Create the HashDB databases in the sizes 4KB, 8KB, 16KB, 32KB, 64KB and 128KB
8. Use the generated list of found files in the NSRL dataset and ingest all the HashDB database with the files
9. Generate the raw datafiles and the statistic from HashDB for further analysis

I obtained three Windows versions for this project:

- Windows 7 x64 Professional SP1 at TU Wien ZID [1]
- Windows 10 Education 32/64-bit (English) at Microsoft Imagine Premium [2]
- Windows Server 2016 Datacenter 64-bit (German) at Microsoft Imagine Premium [3]

[1] https://www.zid.tuwien.ac.at/sts/
[2] https://www.informatik.tuwien.ac.at/msdnaa
[3] https://www.informatik.tuwien.ac.at/msdnaa

All three version were installed with Oracle VMWare on a seperate Virtual Disk Image (VDI) file and than needed to be extracted to a seperate Raw image format (RAW) image for further processing. It lies in the nature of digital forensic work to hash all files before and after working with them, to ensure that no data is changed in the process, so a hash sum has to be generated and I choose a SHA256 hash.

	Command
1	`qemu-img convert -f vdi [vdi image here] -O raw [raw path]`
2	`sha256sum [raw path]`

Table 31: Convert given VDI image to RAW image.

Next I extracted the files of the given image by mounting the image and search for all files with the tool find.

	Command
1	`fdisk -l [raw path]`
2	`mount --read-only -oloop,offset=[calculated offset] [raw path] [mount path]`
3	`find [mount point] -type f -exec sha1sum "{}" + > fileListSha1Sum.txt`

Table 32: Mount given RAW. image and extract all file paths.

For mounting the raw image I had to define a byte offset to mount the desired partition and this offset can be calculated with the information from fdisk. Following source helped me with this issue https://major.io/2010/12/14/mounting-a-raw-partition-file-made-with-dd-or-dd_rescue-in-linux/
In my project the output of the Windows 7 image was

For example I needed the Partition windows7.raw2, I took the start sector 206848 and multiply it with the block size, which is 512. The calculated offset for this image is 105906176.
Before I could proceed, I needed to preapre the NSRL dataset for hfind, by creating a index.

The NSRL dataset only provides MD5 or SHA1 hashes. When the index was created, than I executed *hfind* for every file in the given image and got as result a file with the same amount of lines as the given file which contains the paths to the files. Afterwards I combined both files which will be used for *hashdb*.

	Command
1	`Units: sectors of 1 * 512 = 512 bytes`
2	`Sector size (logical/physical): 512 bytes / 512 bytes`
3	`I/O size (minimum/optimal): 512 bytes / 512 bytes`
4	`Disklabel type: dos`
5	`Disk identifier: 0xf9a1a64d`
6	`Device Boot Start End Sectors Size Id Type`
7	`windows7.raw1 * 2048 206847 204800 100M 7 HPFS/NTFS/exFAT`
8	`windows7.raw2 206848 41940991 41734144 19,9G 7 HPFS/NTFS/exFAT`

Table 33: Output of fdisk.

	Command
1	`hfind -i nsrl-sha1 [path to NSRLFile.txt]`

Table 34: Create index for *hfind*.

Hashdb needs to be set up by creating the database with a defined blocksize for the

	Command
1	`cut -d ' ' -f 1 fileListSha1Sum.txt | xargs -L1 hfind -q [path to NSRLFile.txt] > hashFoundList.txt`
2	`wc -l hashFoundList.txt`
3	`wc -l fileListSha1Sum.txt`
4	`paste -d ';' fileListSha1Sum.txt hashFoundList.txt > nsrlFoundHashes.txt`

Table 35: Execute *hfind* and check if same amount of lines was returned and combine files.

file chunks. After that I used the created nsrlFoundHashes.txt file to filter only those files, which were found in the NSRL dataset, and executed *hashdb* with the path of each file. The command sed 's/ /\\/g' was necessary to replace spaces in the paths with "\", otherwise an error will occur, because the path is not complete.
After the process is finished I got a database wih all sub-file hashes. For the last step I prepared a semicolon-seperated datafile with the information of the nsrlFoundHashes.txt file and extracted information from the files in the image. I used the tool stat to extract:

	Command
1	`hashdb create -b [blocksize] [HashDB directory]`
2	`cut -c 43- nsrlFoundHashes.txt | grep ';1' | cut -d ';' -f 1 | sed 's/ /\\ /g' | xargs -L1 hashdb ingest -s [blocksize] [HashDB directory]`

Table 36: Set up HashDB datbase and ingest file chunks.

- Total size in bytes (%s)
- The size in bytes of each block (%B)
- Number of blocks allocated (%b)
- File name (%n) to check if the extracted information is from the desired file

Also with the last command I inserted the column names at the beginning of the semicolon-sperated datafile.

Besides the file data, I also extracted the hash distribution data of the hashdb database

	Command
1	`sed 's/  /\;/g' nsrlFoundHashes.txt | sed 's/ /\\ /g' > FileListPart1.csv`
2	`cut -d ';' -f 2 FileListPart1.csv | xargs stat -c '%s;%B;%b;%n' > FileListPart2.csv`
3	`paste -d ';' FileListPart1.csv FileListPart2.csv > FileListComplete.csv`
4	`echo 'SHA1HashSum;BashPath;NSRLFound; FileSizeByte;BlockSizeFilesystem;UsedBlocks;Path' | cat - FileListComplete.csv > temp && mv temp FileListComplete.csv`

Table 37: Prepare semicolon-seperated datafile.

with the follwing command.

	Command
1	`hashdb histogram [HashDB directory]|`

Table 38: Print HashDB database hash distribution.

The generated HashDB databases are provided in compressed tar format. The full list can be found in the appendix.

3.1 Obstacles

For the first attempt, I used the tool *fiwalk* to extract the metadata information of the given image.

	Command
1	`fiwalk -Xfiwalkout.xml -I [raw path]`

Table 39: Create fiwalk Extensible Markup Language (XML) file output, without NTFS system files.

The problem was, that fiwalk also extracted the metadata information of all directories in the given image, so it would distort the result of the analysis afterwards. An example is showned in 310.

	Fiwalk output
1	`<fileobject>`
	`...`
2	`<filename>$Extend/$RmMetadata/.</filename>`
	`...`
3	`<filesize>336</filesize>`
	`...`
4	`<hashdigest type='md5'>`
	`dd269177af94f4d25351e54e519b1468`
	`</hashdigest>`
5	`<hashdigest type='sha1'>`
	`a5d4b376d8d3c1945520614b9d5a20cb84d5082e`
	`</hashdigest>`
	`...`
6	`</fileobject>`

Table 310: Fiwalk output directory example. Filesize in bytes

Important for this project was the extraction of known files, found in the NSRL dataset, so I had to change my approach and startet mounting the image and generated a list of all files within the given image.

Additionally I first used the tool *hfind* exclusively with the command -f, which takes a file of hashes and searches the NSRL dataset for every given hash in the file. For each found hash *hfind* will output all matched files. See 311.

	Hfind output
1	`hfind NSRL\_DVD/RDS\_255_modern/NSRLFile.txt "f50991f041c04514d3a3610c2593f694a2fbc7e5"`
2	`2221`
3	`2231`
4	`FL_wizardProviderInfo_ascx_102277 _____A64.3643236F_FC70_11D3_A536_0090278A1BB8`
5	`FL_wizardProviderInfo_ascx_102277_____ I64.3643236F_FC70_11D3_A536_0090278A1BB8`
6	`FL_wizardProviderInfo_ascx_102277 _____X86.3643236F_FC70_11D3_A536_0090278A1BB8`
7	`wizardProviderInfo.ascx`
8	`wizardproviderinfo.ascx`
9	`wizardproviderinfo_ascx_amd64`
10	`wizardproviderinfo_ascx_x86`

Table 311: Hfind output example - shorten

For further processing, I needed just an indicator if a file was found or not, so I used *hfind* with the command -q as above mentioned in Implementation and methods.
I startet working with the image of Windows 7. After the first verfication of the output data I figured out, that over 50% of the files wasn't found in the NSRL dataset but I was expecting, that 75-80% should be found within the dataset. I revised my approach afterwards and used it in the final version of this project.

The next obstacle affected *hashdb*. During my work with *hashdb* I encountered a compiling problem. Arch Linux is a rolling release distribution, which makes full system updates very easy and keeps the operational system up-to-date. Since an update *hashdb* couldn't compile any more and my first examination of this error ended withoutresult, therefore I raised an issue at *Github* [4], which was answered by Simson Garfinkel [5], who implemented *bulk_extractor* and does research in the field of digital forensics. His work on *hashdb* and *hash-bases carving* provided the foundation for this project.
After some conversation, there was unfortunately still no solution for my problem. In the meantime I examined the problem on my own and figured out, that with one system update *OpenSSL* was updated from version 1.0 to version 1.1.0 and *hashdb* is using the

[4] https://github.com/NPS-DEEP/hashdb/issues/6
[5] https://simson.net

function *SSL_library_init* which was now deprecated [6] and replaced with the function *OPENSSL_init_ssl* [7].
To get *hashdb* with *OpenSSL* version 1.1.0, I had to replace the function in the configuration file of the installation [8] to:

	configure.ac
206	`AC_CHECK_LIB([ssl],[OPENSSL_init_ssl], [],[AC_MSG_ERROR([Could not find ssl library])])`

Table 312: HashDB configure.ac file line 206

My solution to this issue is only fixing the problem with *OpenSSL* version 1.1.0. When I provided my solution to *Github* Mr. Garfinkel, he also pointed out, that *hashdb* won‘ t work with *OpenSSL* version 1.0, but shortly after he provided me a solution for both versions.

[6] https://www.openssl.org/docs/man1.1.0/ssl/SSL_library_init.html
[7] https://www.openssl.org/docs/man1.1.0/ssl/OPENSSL_init_ssl.html
[8] https://github.com/NPS-DEEP/hashdb/blob/master/configure.ac#L206

CHAPTER 4

Data analysis

The datasets, graphics and Tableau workbook can be downloaded under following link https://s3-eu-west-1.amazonaws.com/dtichy-bachelorthesis/Graphics+and+Data.7z

The first graphic shows how many files where found in the NSRL dataset. With 81.592 files Windows 10 has more files than Windows 7 (66.481) or Windows Server 2016 (68756) and with 98% the highest found rate of all other analyzed operational systems.

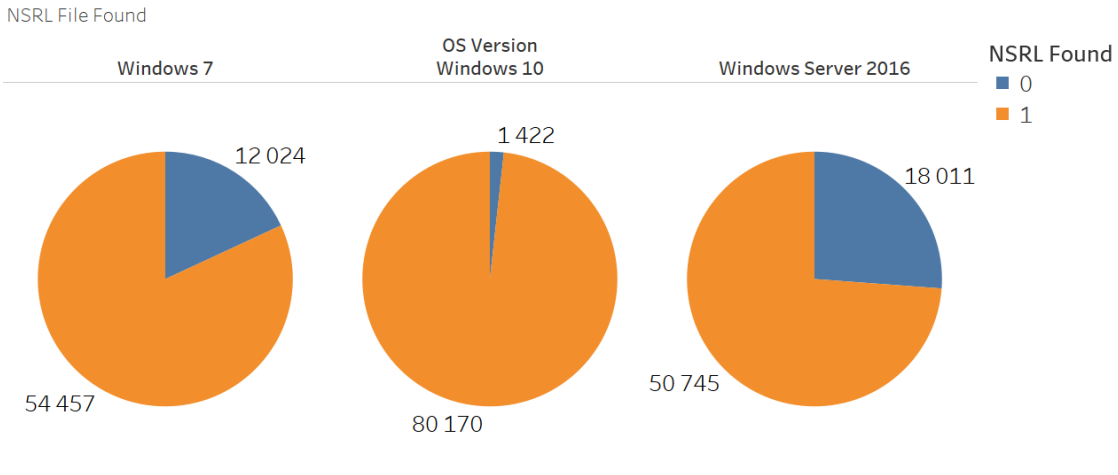

Figure 41: OS files found in NSRL

The next three graphics show the file sizes, clustered by 4KiB, 8KiB, 16KiB, 32KiB, 64KiB, 128KiB, 256KiB and greater than 256KiB and divide them by NSRL found status. Windows 7 and Windows Server 2016 has a simliar distribution of files. It is noticable that Windows 10 has more files which are less or equal 4KiB in size, than Windows 7 or Windows Server 2016.

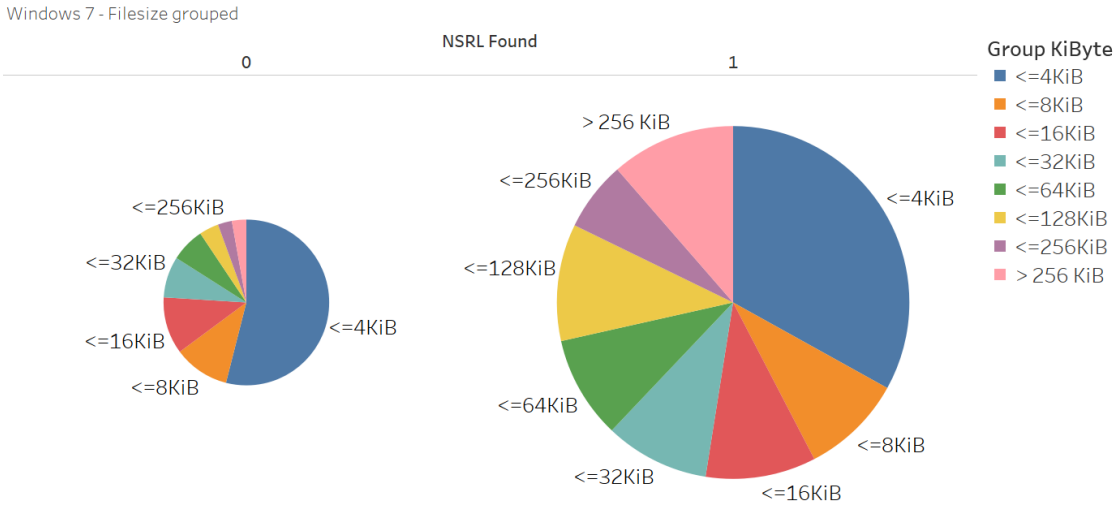

Figure 42: Windows 7 size of files clustered

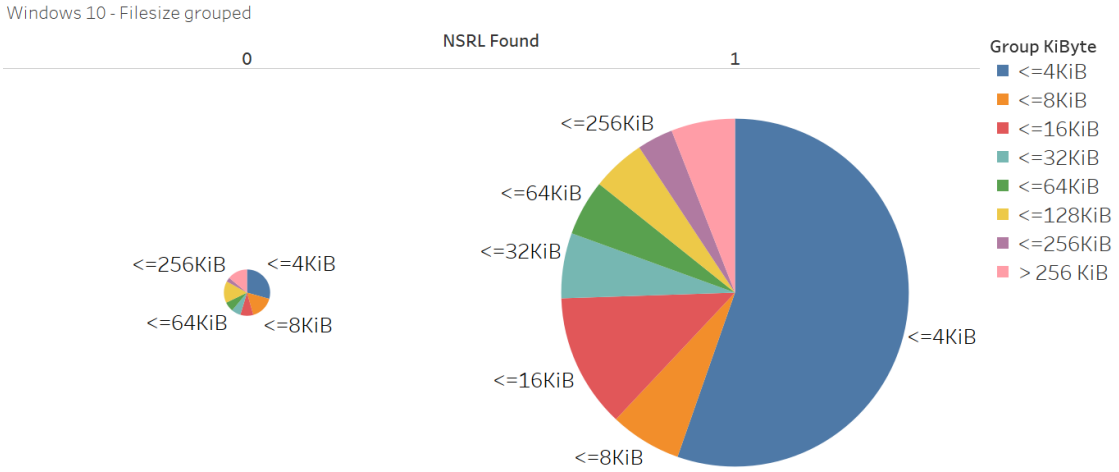

Figure 43: Windows 10 size of files clustered

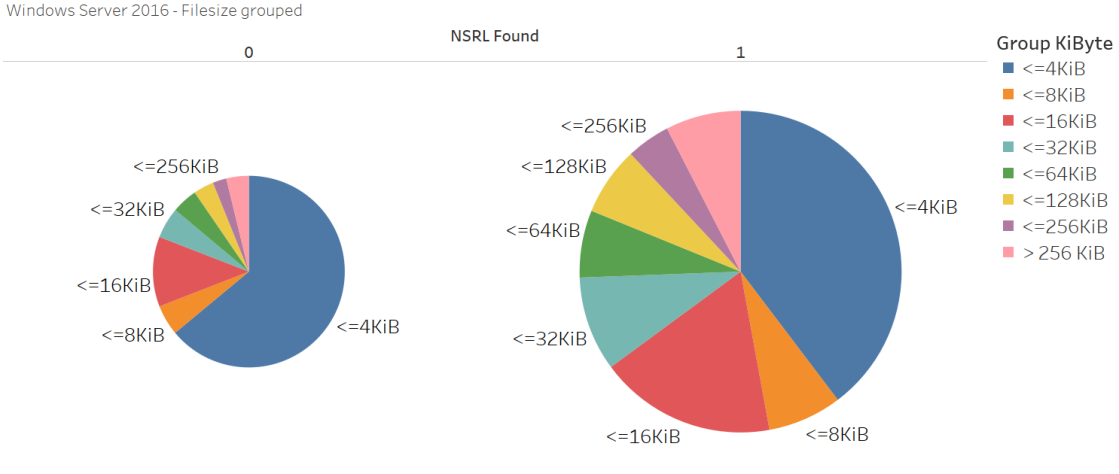

Figure 44: Windows Server 2016 size of files clustered

Additionally I evaluated how often file hashes occured overall in the given images for all hashes, which were found in the NSRL dataset. For example, in Windows 7 16.838 Hashes were distinct, so each hash value was unique, 15.689 Hashed were found twice and 2.906 Hashes occured three times. Windows 7 and Windows 10 has much more hashes which occured twice then Windows Server 2016.

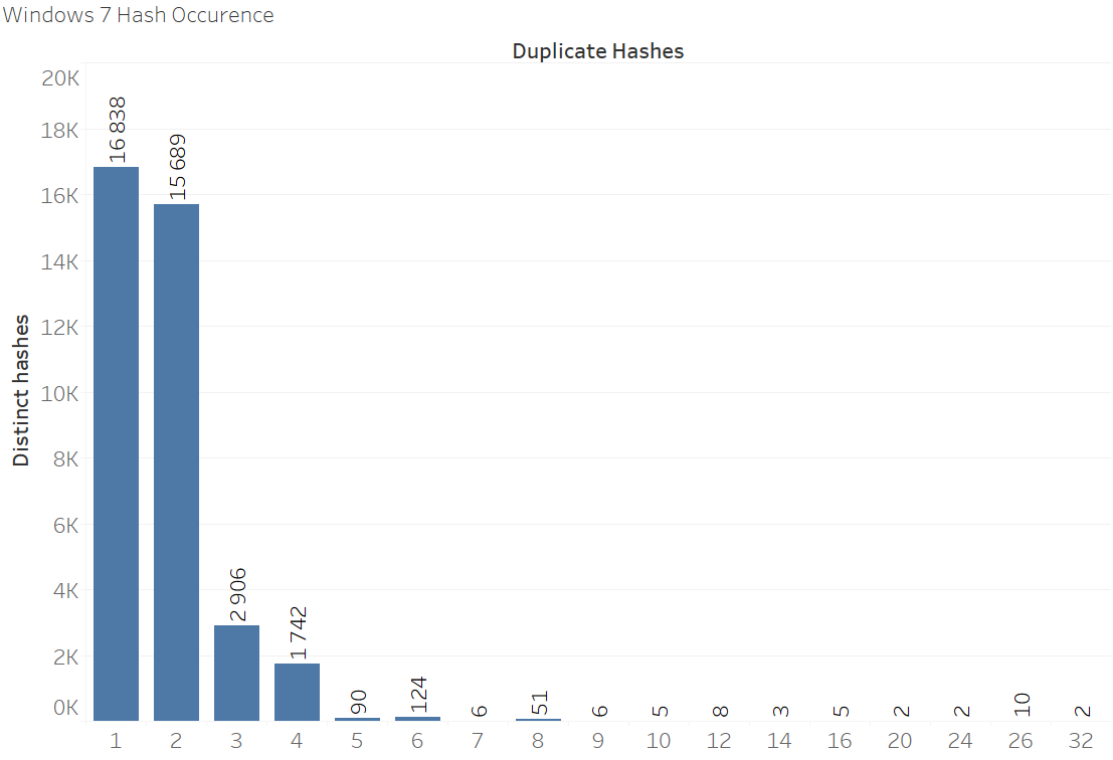

Figure 45: Windows 7 Hash Occurence for NSRL found files

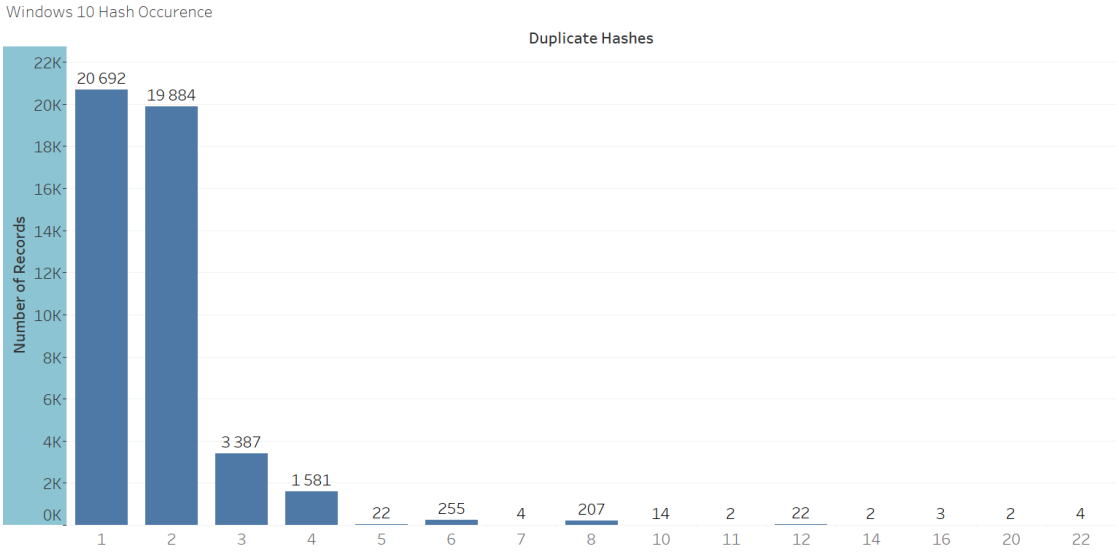

Figure 46: Windows 10 Hash Occurence for NSRL found files

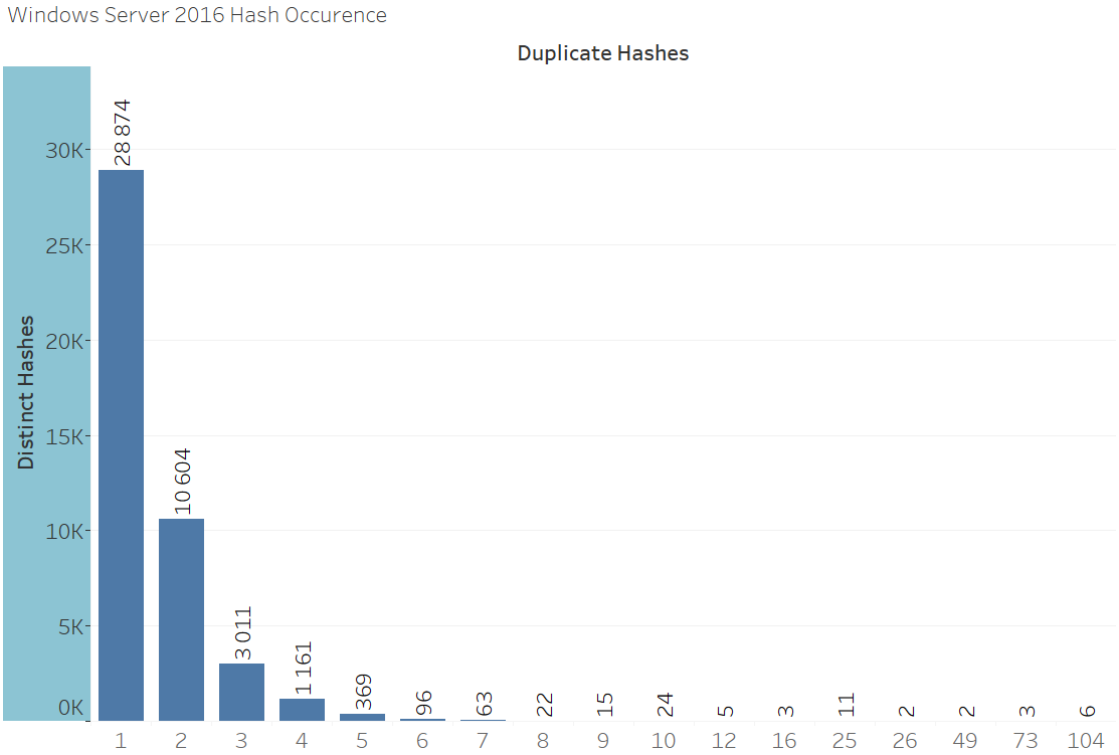

Figure 47: Windows Server 2016 Hash Occurence for NSRL found files

At least I added the output of the *hashdb* histogram. I splited the graphics into the used blocksizes 4k, 8k, 16k, 32k, 64k and 128k. A trivial fact can be seen in all three operational systems: The higher the blocksize is chosen for sub-file hashing, the lesser duplicate hashes occur, because lesser hashes will be produced. All graphics contains only the top 22 duplicates.

Series of 4k blocksize.

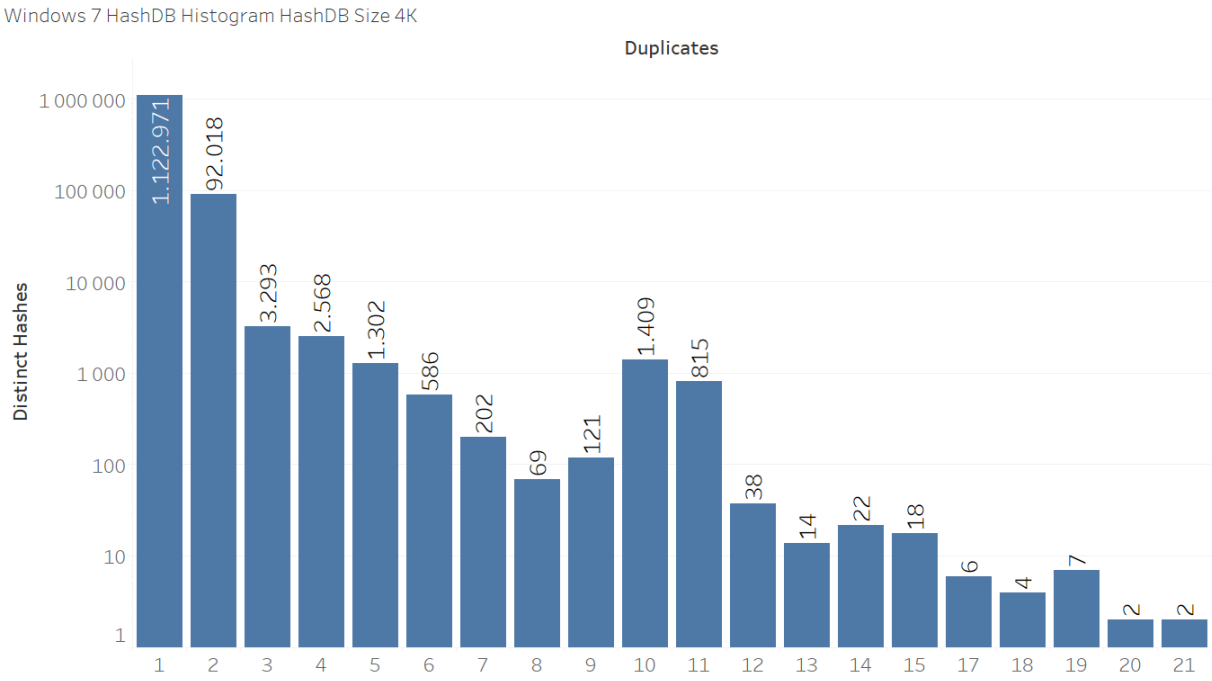

Figure 48: Windows 7 HashDB Histogram 4k blocksize

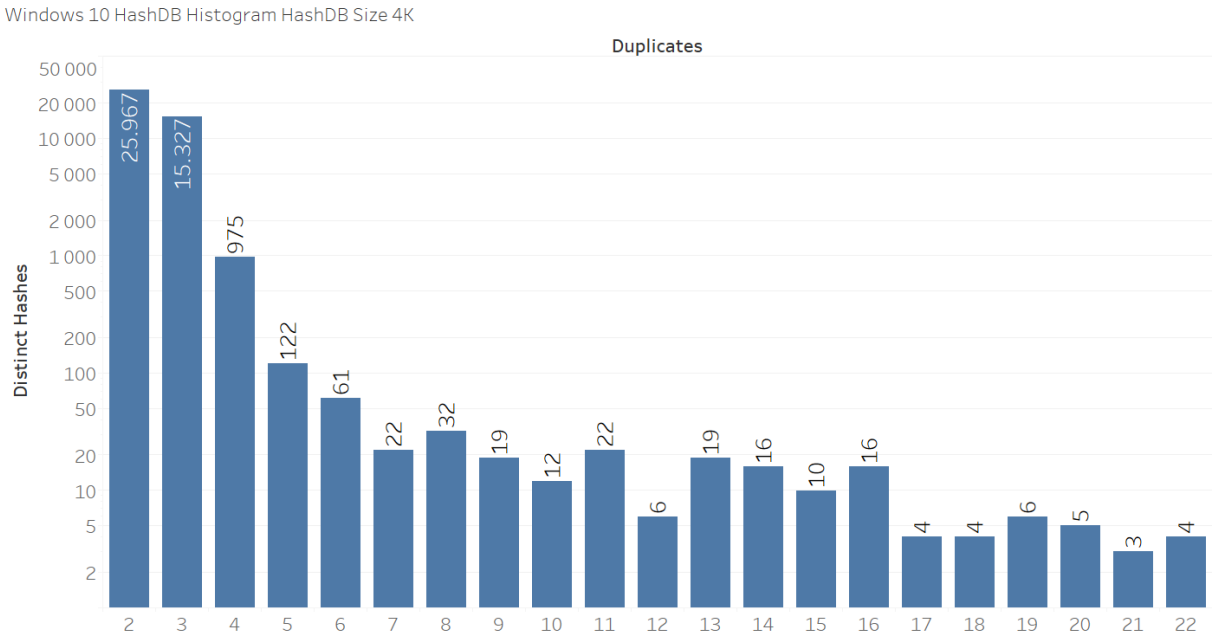

Figure 49: Windows 10 HashDB Histogram 4k blocksize

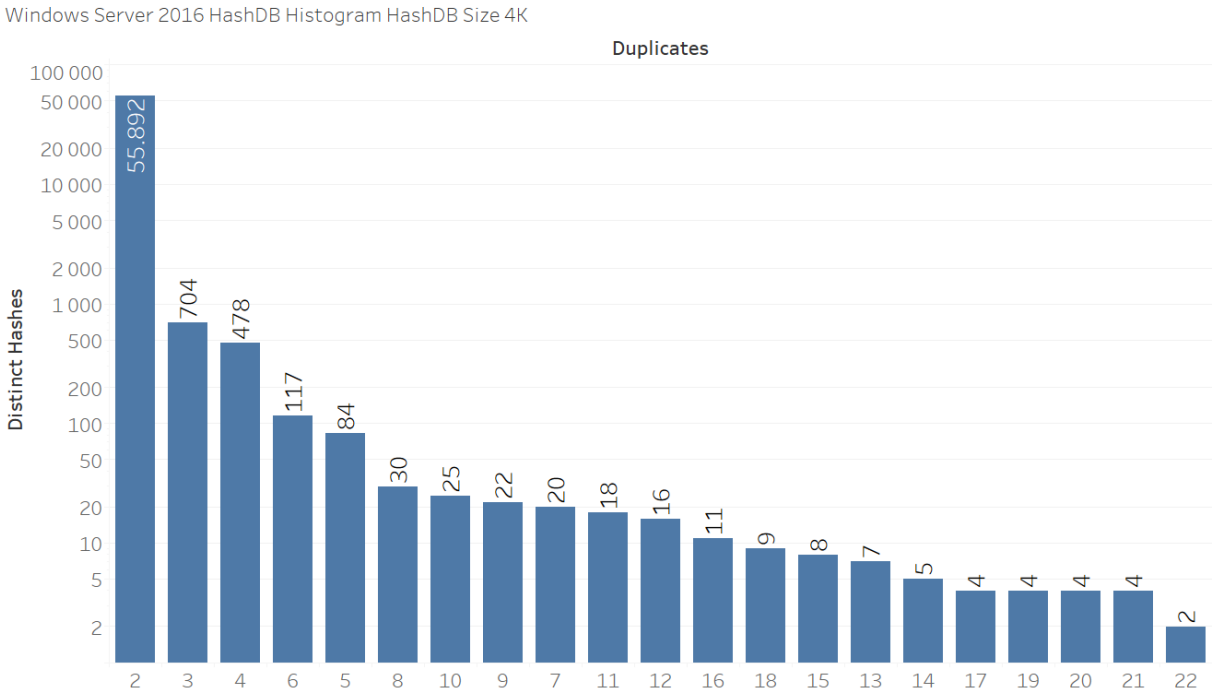

Figure 410: Windows Server 2016 HashDB Histogram 4k blocksize

Series of 8k blocksize.

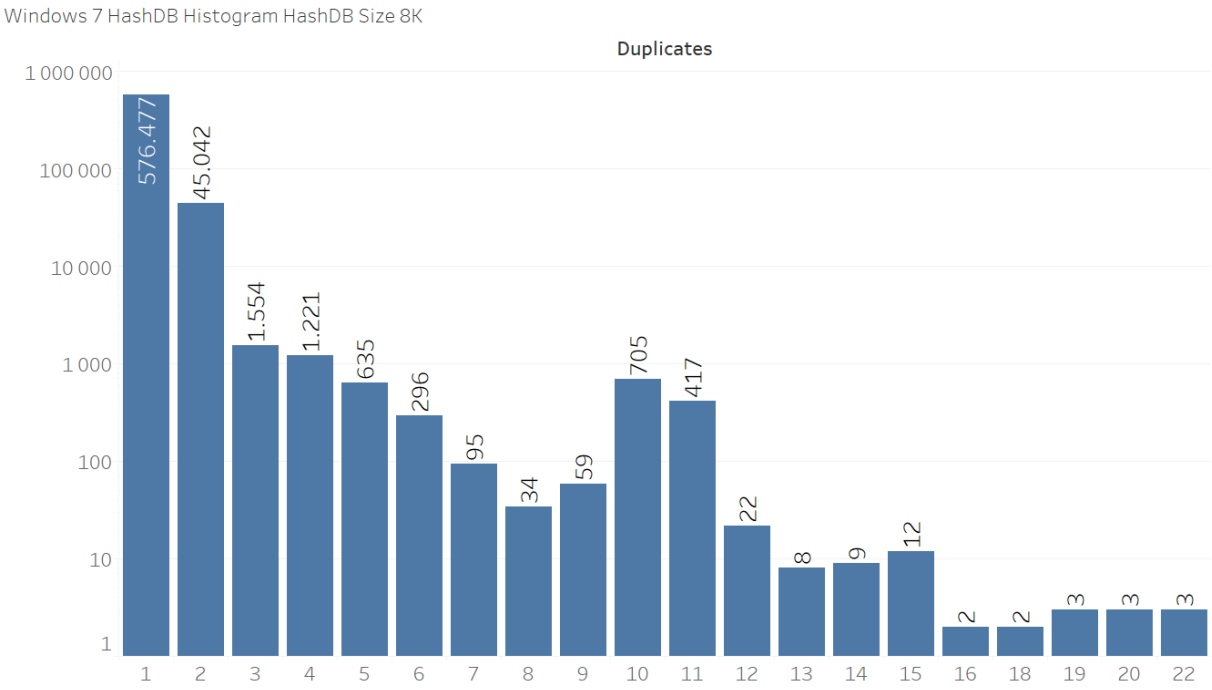

Figure 411: Windows 7 HashDB Histogram 8k blocksize

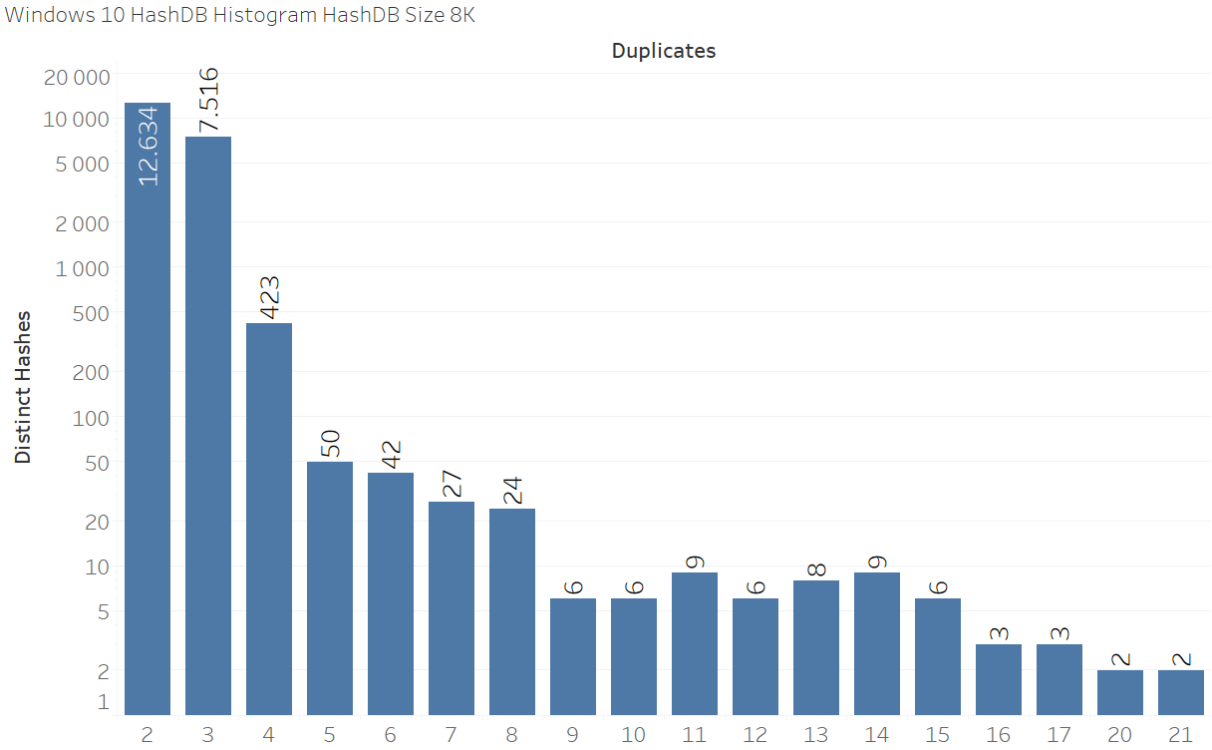

Figure 412: Windows 10 HashDB Histogram 8k blocksize

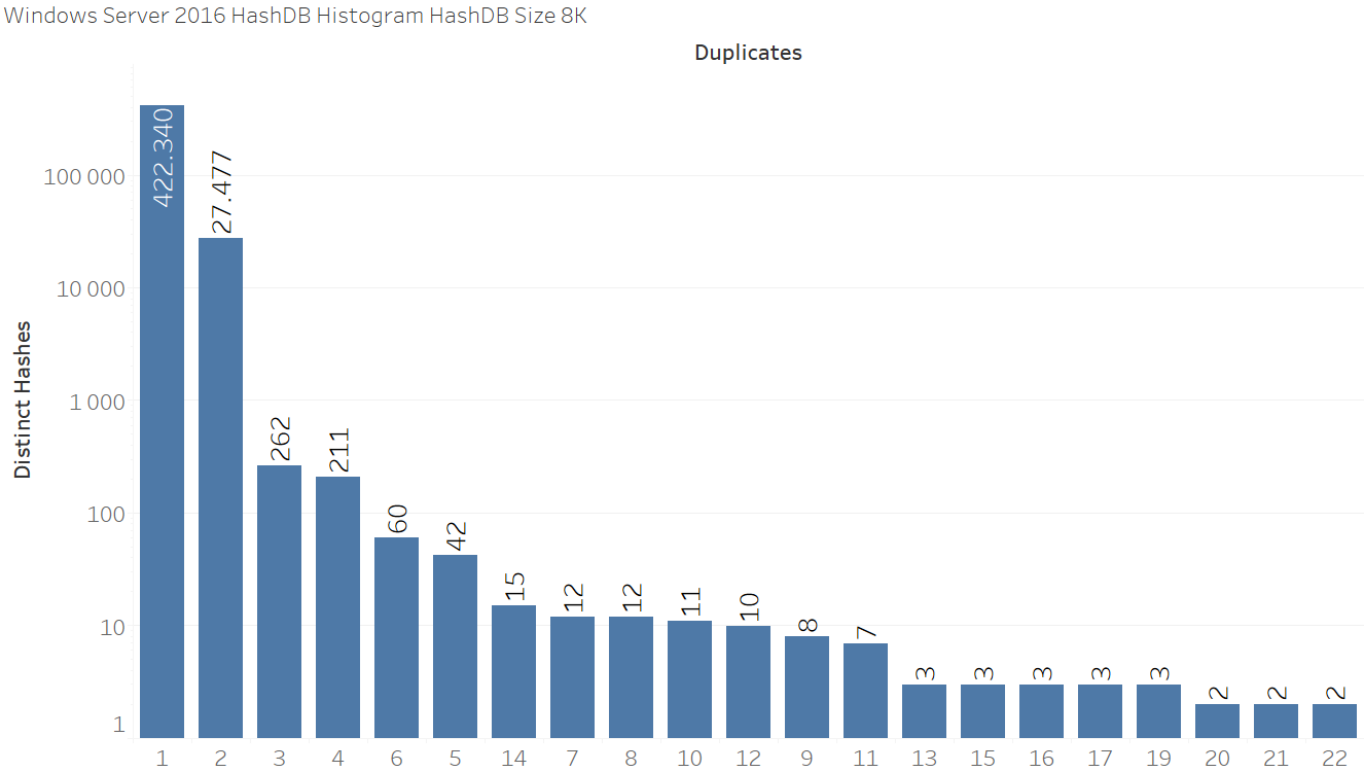

Figure 413: Windows Server 2016 HashDB Histogram 8k blocksize

Series of 16k blocksize.

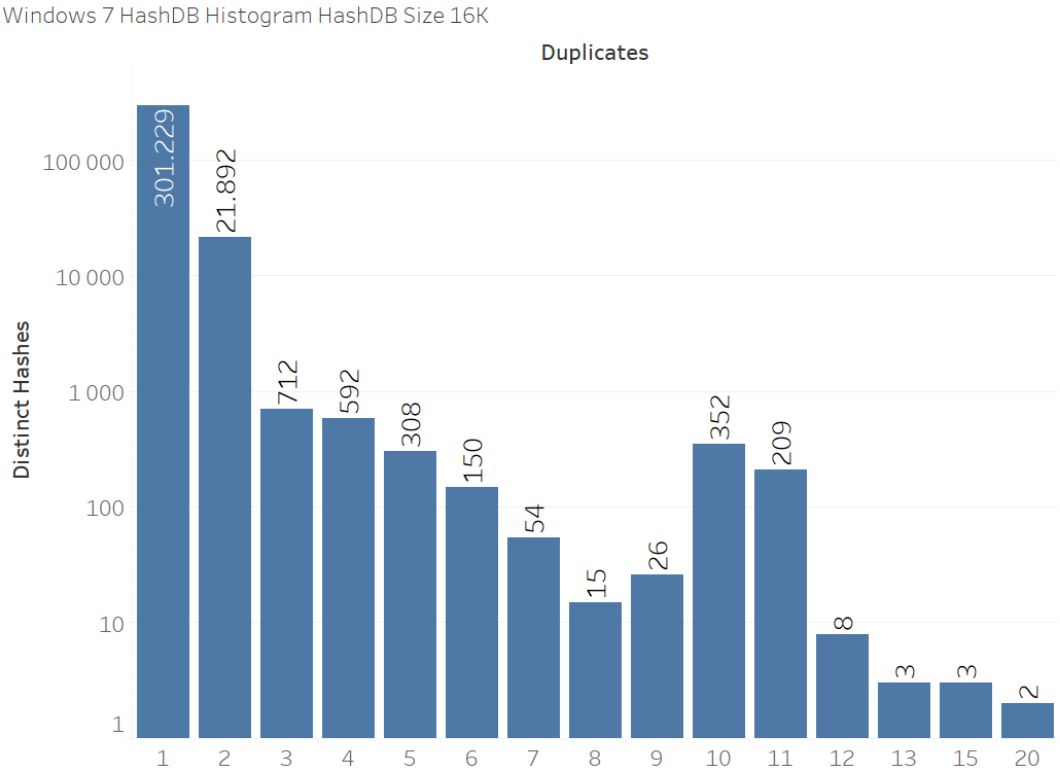

Figure 414: Windows 7 HashDB Histogram 16k blocksize

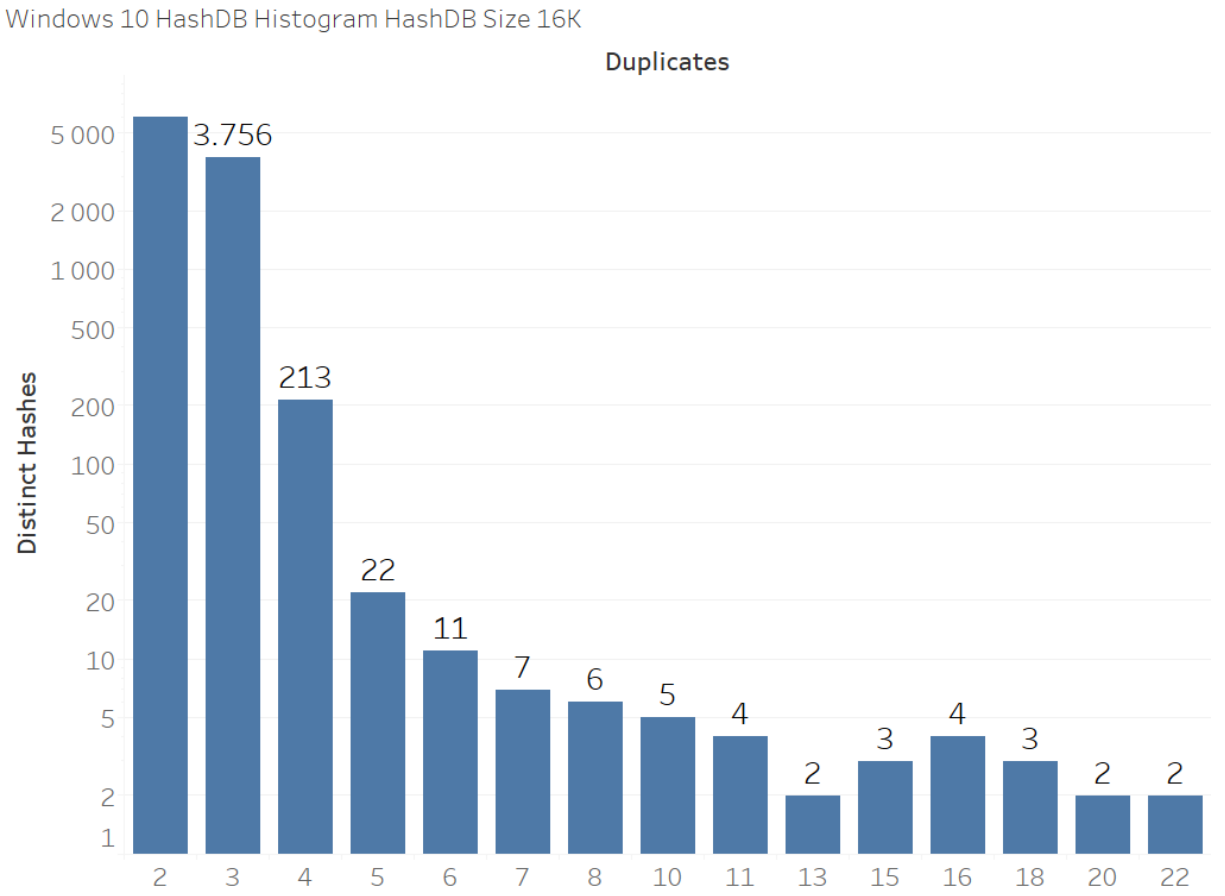

Figure 415: Windows 10 HashDB Histogram 16k blocksize

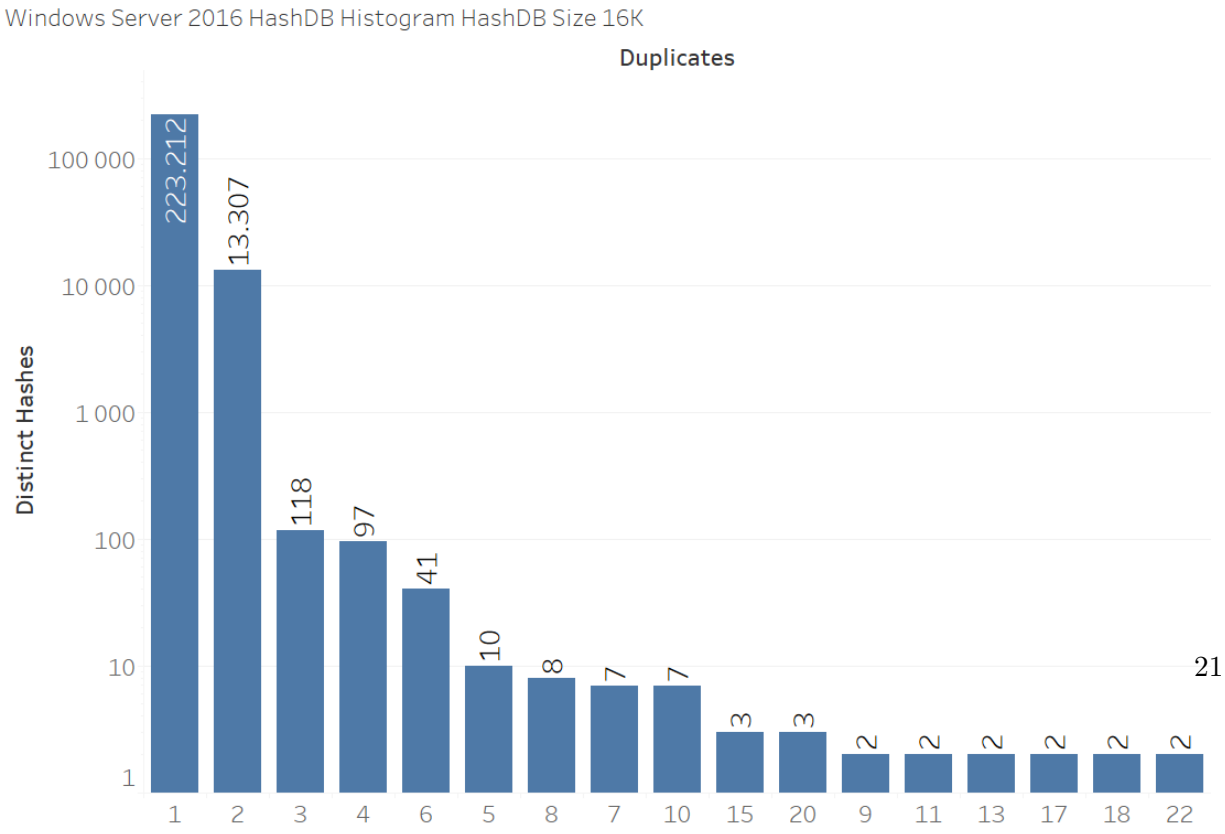

Figure 416: Windows Server 2016 HashDB Histogram 16k blocksize

Series of 32k blocksize.

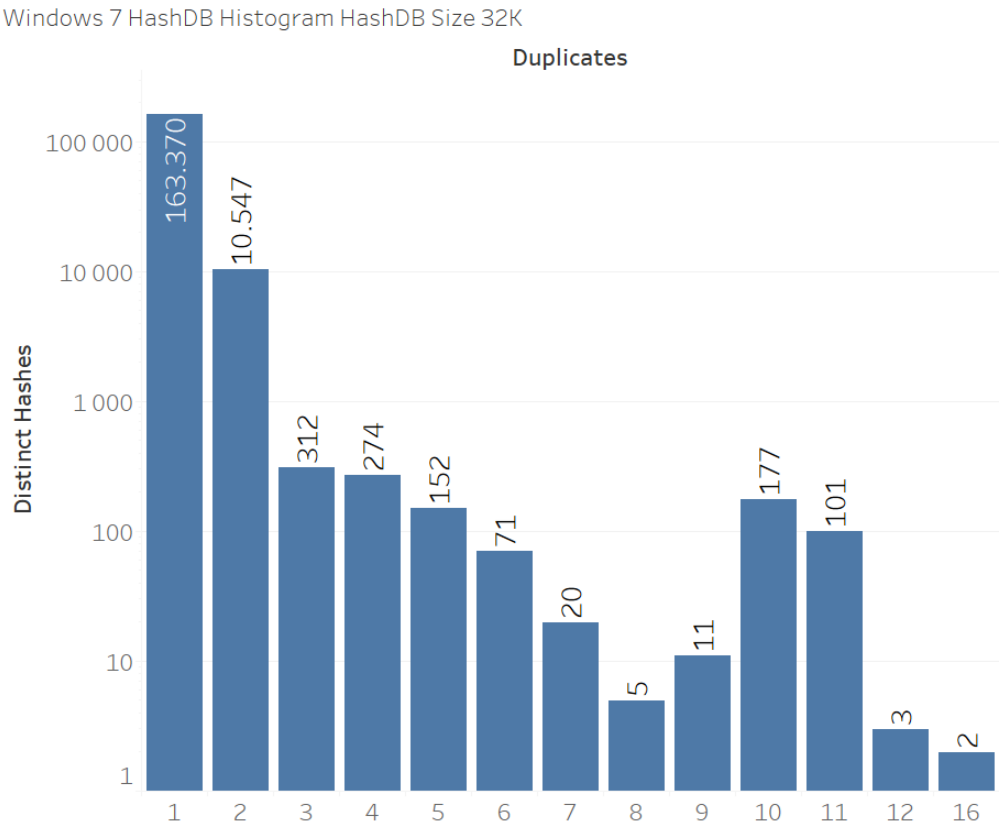

Figure 417: Windows 7 HashDB Histogram 32k blocksize

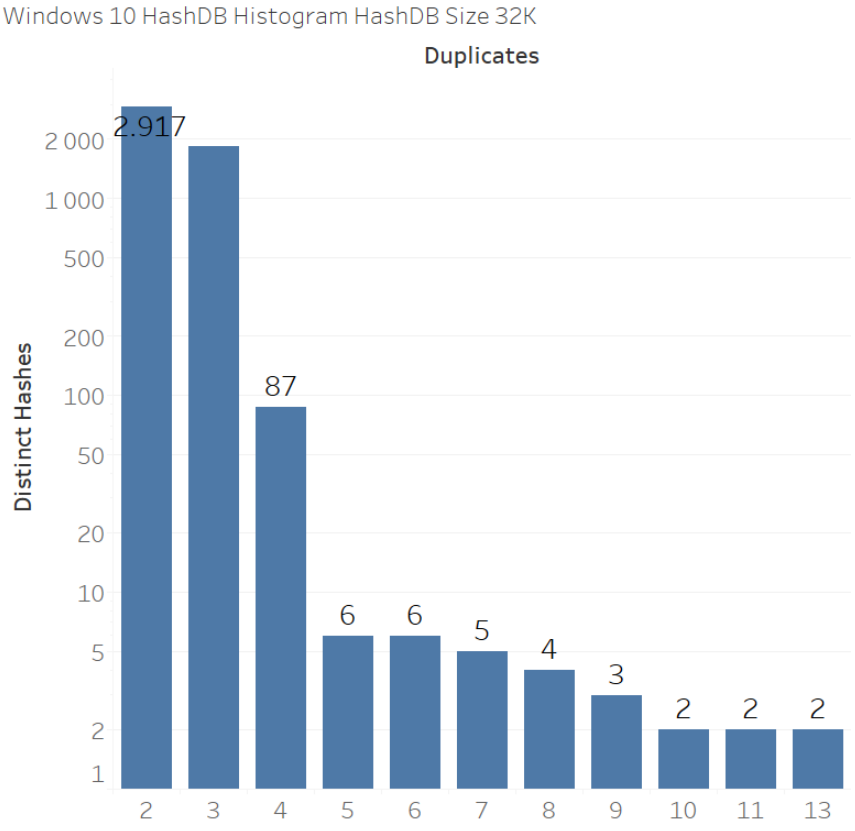

Figure 418: Windows 10 HashDB Histogram 32k blocksize

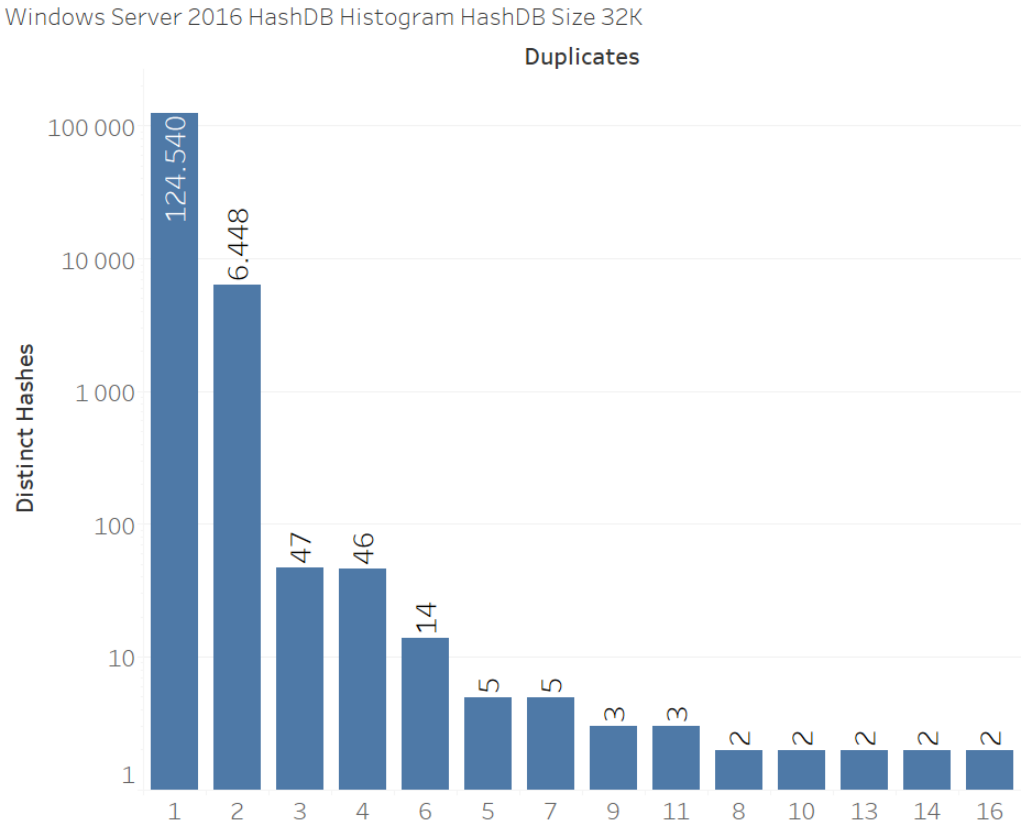

Figure 419: Windows Server 2016 HashDB Histogram 32k blocksize

Series of 64k blocksize.

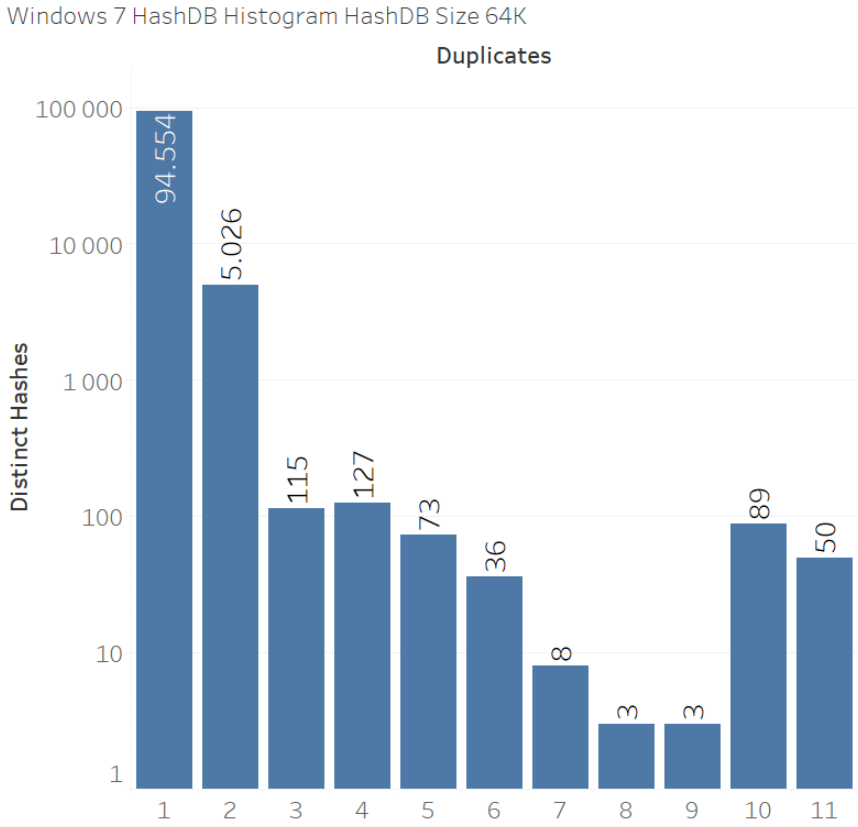

Figure 420: Windows 7 HashDB Histogram 64k blocksize

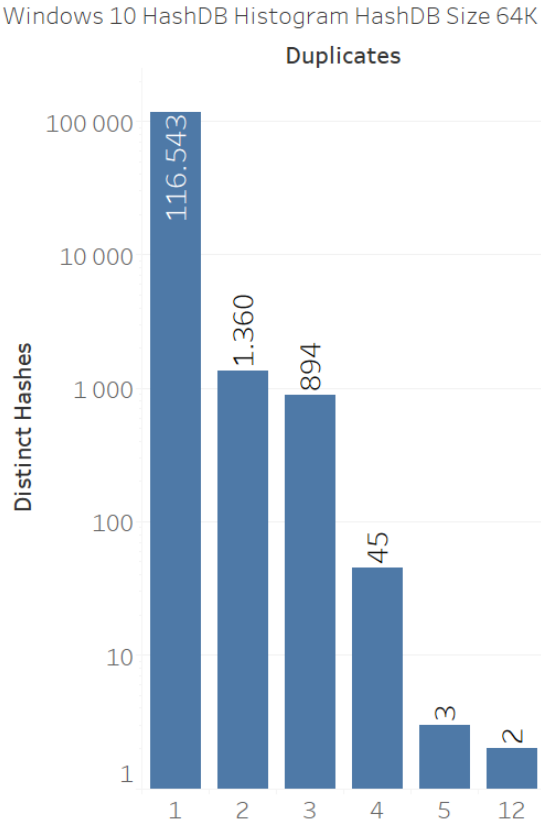

Figure 421: Windows 10 HashDB Histogram 64k blocksize

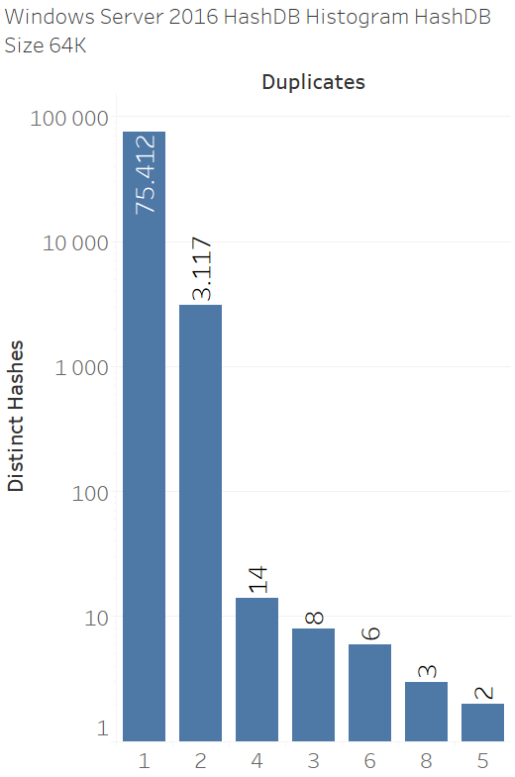

Figure 422: Windows Server 2016 HashDB Histogram 64k blocksize Series of 128k blocksize.

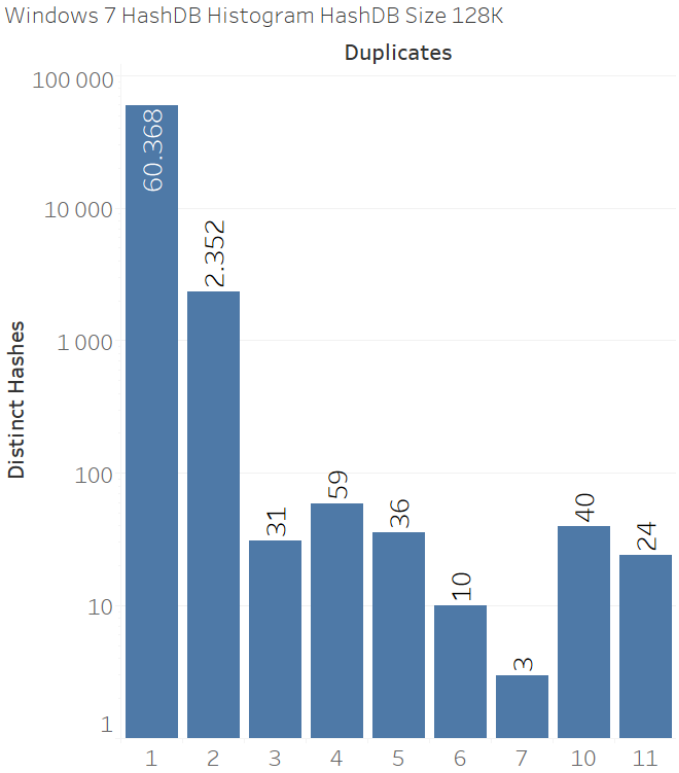

Figure 423: Windows 7 HashDB Histogram 128k blocksize

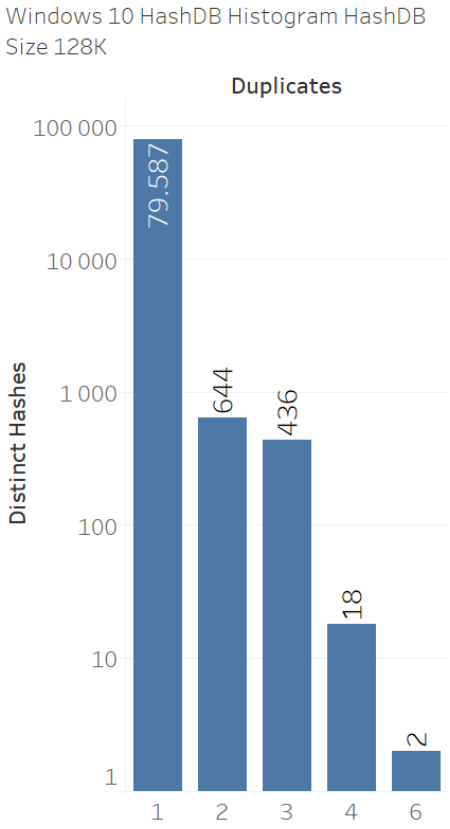

Figure 424: Windows 10 HashDB Histogram 128k blocksize

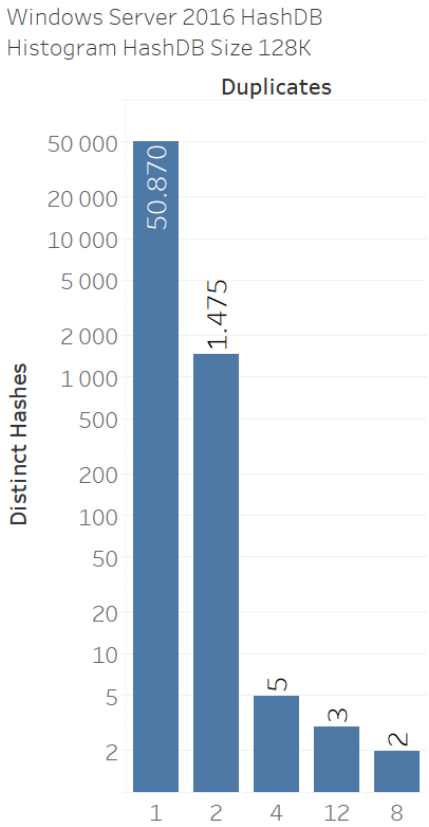

Figure 425: Windows Server 2016 HashDB Histogram 128k blocksize

CHAPTER 5

Conclusion

First of all, using Windows, as operational system as base for this work, has the most impact on forensic analysis, due to the distribution of Windows and especially Windows 7 at the moment. Surely the process can be used on all known operational systems, but I would prefere to then use only files, which are also included in the NSRL dataset. To the question, if enough files are found in the NSRL dataset, I expected, that at least 2/3 of all files are also included in the NSRL dataset. As seen in Figure 4.1 there are even more then enough files for all three operational systems and it would fasten the analysis of a forensic image, which includes one of the used operational systems. During my work on the first image, I minimized the steps as far as I could, seen in the chapter "Implementation and methods". When I was preparing the data for the last 2 images, the prepared statements has fasten up the work immense. Still, I had some checks to do between the checks, especially for preparing the CSV files, but creating a script in bash, or using the described process in a Python module, would help to automate the whole procedure.

CHAPTER 6

Discussion

First of all I learned that the open-source tools like *hashdb* or the tools provided in *sleuthkit* are very powerful for forensic analysis, or preparing data for analysis, but in my opinion most of the documentary was quite minimalistic. It requires much time to test the tools and find some walktroughs in addition to the documentation which supported me.

I quickly realized, that the build-in unix tools form the best base enviroment around the tools itself, to prepare the data for *hashdb*, which also makes the whole project easier to set up on an own unix enviroment.
The compiling problem, which I had with *hashdb* not only gave me the opportunity to talk to Simson Garfinkel, it also had a learning effect since I examined some of the source code of *hashdb*.

The outcome of the data analysis were in some points unexpected, like the distribution of the found files in the NSRL dataset. I was expecting, that around 75-80% of the files within the operational systems were also in the NSRL dataset, but nearly all files within Windows 10 where found. On the other hand, lesser files from Windows 7, the older operational system, were found in the NSRL dataset, but still the percentage of the found files where in the expected range. Also, since the NSRL dataset is using the SHA1 hash algorithm and the majority of the files has a size less then 4KiB, I was expecting more hashes which were found 3 or more times.

CHAPTER 7

Future work

Here I will mention some points, which can be done afterwards, based on this project. Primary, as mentioned, a script for preparing the HashDB databases could automate the whole procedure, for faster creating HashDB databases. I also think, that a bigger store of prepared HashDB databases can be build and provided for forensic analysis, like in the project Peekatorrent. For that, more operational systems should be taken and also maybe regular updates of existing HashDBs could be provided, because the NSRL dataset will be updated regulary and maybe more files could be found in the dataset, so that the HashDB databases contain more data.
Further it should be extended for operational devices used on smartphones, like android, to support mobile device forensic analysis, but for that, the NSRL dataset should be tested against an image of a smartphone, if enough files are included.
Since all the questions of this work has been answered positive, I would recommend to create *hashdb* databases of as many modern operational systems as possible and provide the databases to all forensic teams, to fasten up their work. Since the update of the databases could be too much for one organisation, I would recommend to put this project up as an open-source project, so the work can be distributed to many developers, who are willing to participate to this work.

CHAPTER 8

Used tools

Enclosed a list of the tools, which were used for this project.

- Arch Linux - linux 4.12.8-2
- hashdb 3.l.0
- fdisk from util-linux 2.30.1
- find 4.6.0
- cut 8.27
- paste 8.27
- sed 4.4
- xargs 4.6.0
- hfind from sleuthkit 4.4.2-1
- fiwalk from sleuthkit 4.4.2-1
- qemu-img 2.9.0
- Oracle VirtualBox GUI 5.1.26 r117224
- Graph visualization Tableau Desktop Professional 10.3.2 [1]

[1](https://www.tableau.com/)

Appendix

HashDB Name	Hashes	**tar Size**	**Download Link**
Win7_4096B_HashDB.tar	1375280	241MB	Win7_4096B_HashDB.tar
Win7_8192B_HashDB.tar	698672	159MB	Win7_8192B_HashDB.tar
Win7_16384B_HashDB.tar	360101	111MB	Win7_16384B_HashDB.tar
Win7_32768B_HashDB.tar	191429	87MB	Win7_32768B_HashDB.ta
Win7_65536B_HashDB.tar	107752	83MB	Win7_65536B_HashDB.tar
Win7_131072B_HashDB.tar	66393	70MB	Win7_131072B_HashDB.tar
Win10_4096B_HashDB.tar	1352262	305MB	Win10_4096B_HashDB.tar
Win10_8192B_HashDB.tar	695427	209MB	Win10_8192B_HashDB.tar
Win10_16384B_HashDB.tar	366347	161MB	Win10_16384B_HashDB.tar
Win10_32768B_HashDB.tar	203369	137MB	Win10_32768B_HashDB.tar
Win10_65536B_HashDB.tar	122382	125MB	Win10_65536B_HashDB.tar
Win10_131072B_HashDB.tar	82325	125MB	Win10_131072B_HashDB.tar
WinServer2016_4096B_HashDB.tar	948989	158MB	WinServer2016_4096B_HashDB.tar
WinServer2016_8192B_HashDB.tar	485828	110MB	WinServer2016_8192B_HashDB.tar
WinServer2016_16384B_HashDB.tar	253456	86MB	WinServer2016_16384B_HashDB.tar
WinServer2016_32768B_HashDB.tar	138866	74MB	WinServer2016_32768B_HashDB.tar
WinServer2016_65536B_HashDB.tar	82129	74MB	WinServer2016_65536B_HashDB.tar
WinServer2016_131072B_HashDB.tar	53974	68MB	WinServer2016_131072B_HashDB.tar

List of Figures

List of Tables

Bibliography

[Car05] Brian Carrier. *File System Forensic Analysis*. Addison-Wesley Professional, 2005.

[SLGa14] Michael McCarrin b Simson L. Garfinkel a. Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb. *Digital Investigations*, 14(7):95–105, 2014.

[SN16] Edgar Weippl Sebastian Neuner, Martin Schmiedecker. Peekatorrent: Leveraging p2p hash values for digital forensics. *Digital Investigations*, 18(7):149–156, 2016.