MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Disk sector content analysis and visualization

Bachelor's Thesis

JAKUB MALOŠTÍK

Brno, Spring 2022

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Disk sector content analysis and visualization

Bachelor's Thesis

JAKUB MALOŠTÍK

Advisor: Ing. Milan Brož, Ph.D.

Department of Computer Systems and Communications

Brno, Spring 2022

FACULTAS ARTIS INFORMATICÆ • UNIVERSITAS MASARYKIANA •

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Jakub Maloštík

Advisor: Ing. Milan Brož, Ph.D.

Acknowledgements

These are the acknowledgements for my thesis, which can span multiple paragraphs.

Abstract

This is the abstract of my thesis, which can span multiple paragraphs.

Keywords

keyword1, keyword2, ...

Contents

List of Tables

List of Figures

Introduction

Disks (e.g., hard drives, SSDs, Flash drives) are usually divided into atomic parts named sectors, which are represented as blocks in the software layer. Sectors store a fixed amount of data, usually 512 bytes and 4KiB, but other sector sizes can be used. Sectors may contain partition tables, file system information, files or be empty.

Some of the possible contents may contain specific byte patterns which can be analyzed and used to identify the type of content stored in the sector. When a byte pattern is not present, sector content can be analyzed for entropy to estimate whether it is encrypted. A good way to get an idea about which parts of the disk are encrypted and where filesystem data is stored is to visualize the data. This visualization will allow humans to distinguish between different data encryption methods such as filesystem-level and full-disk encryption and even uncover faulty encryption. Visualizing can also be very useful as an illustration while teaching.

The utility introduced in this bachelor's thesis analyzes the sectors of a user-specified size of a provided disk image and visualizes the result using the Pillow Python library. The utility is also easily extensible by other output methods.

The text of this thesis is structured into five chapters. Chapter number one explains the foundations of the thesis and examines prior work. Chapter number two lists some byte patterns of sectors and discusses algorithms for their detection. Chapter number three discusses algorithms used to calculate entropy and possible issues with their accuracy. Chapter number four discusses ways of visualization and their advantages and disadvantages. The last chapter concludes with an evaluation of the resulting utility.

The resulting utility is available on GitHub[1] under the MIT License.

[1] https://github.com/malon43/entropy-visualization

1 Prior work

This review focuses on works on the topics of block detection, entropy calculation, and ways of visualization.

1.1 Analysis

This section describes parts of analyzing the disk sectors.

1.1.1 Block Patterns

Each disk is divided into tens, even hundreds of millions of sectors. Each disk sector stores some data. Sectors of empty new drives would be mostly initialized with a pattern of zeroes, except for partitioning tables and file system metadata.

Most recent drives use 4KiB sized sectors, also known as Advanced Format, but still provide backward compatibility with older systems which expect 512B sector size with 512B sector size emulation.[1]

Sector byte pattern is a specific configuration of bytes, which would indicate what this sector is used for. For example, a repeated pattern of byte x00 often signalizes that this sector has not been used yet, that the blocks have been freed by the TRIM command. The TRIM command is used by the software to inform the drive which sectors no longer contain user data in order to increase performance.[2] Or, bytes x55xAA at the end of the sector would signalize a block containing master boot record (MBR). However, while in many cases, analysis for positions of bytes is not as time-intensive as analysis of randomness or single-byte patterns, multiple problems show up:

- Testing for many positions and byte configuration will add up.
- Files with magic bytes may be contained in the first sector where the file is stored, but there is no easy way of telling whether the file simply ends, continues on the next sector, or is placed in a completely different sector.
- If the file is unencrypted, it will mostly get picked up by the randomness analysis.

Most works focusing on detecting patterns of bytes on sectors[3, 4] do it through the lens of forensic analysis and use the filesystem metadata in combination with magic bytes of files to allow the user to find information faster. These, while up an abstraction layer from what this thesis focuses on, can provide beneficial information when identifying common patterns of entire sectors or repeating portions of bytes in a single sector.[3]

1.1.2 Randomness

In order to properly classify all disk sectors, one cannot rely exclusively on byte patterns since files can span multiple sectors and can even be encrypted. In this case, it is possible to check the predictability of byte values or even of single bits.

In order to precisely differentiate random data, the provided samples would need to be in the order of gigabytes, which is far from the provided 512 or 4096 Bytes. However, we can at least get an estimate using the techniques described in this subsection.

Entropy

Shannon's entropy calculates the amount of information in bits provided by each byte value in the sector.[5] For example, the entropy of 8 bits means that every byte value is contained the same number of times (i.e., exactly $\frac{s}{256}$ times). Whereas the entropy value of 0 means that only a single byte value is contained and is repeated through the whole sector. Shannon's entropy can be calculated using:

$$H(S) = -\sum_{i=0}^{255}(P(x_i)\log_2(P(x_i)))$$

Where $P(x_i)$ represents the probability of byte value i (i.e., number of times value i appears in the sector divided by the number of all bytes in the sector). Which can be then normalized:

$$\mu(S) = \frac{H}{H_{max}} = -\frac{1}{8}\sum_{i=0}^{255}(P(x_i)\log_2(P(x_i)))$$

Where s is equal to the sector size in bytes. Normalized Shannon's entropy ranges from 0, the least random (a single repeated byte value), to 1, the most random (every byte value is contained in the sector an equal amount of times). Using this value, one can estimate whether the sector contains encrypted data.

However, multiple problems arise when using Shannon's entropy. There is no simple line where all sectors with a higher entropy are encrypted, and all with lower entropy are not. That means that most sectors containing compressed file formats like videos, jpeg images, or zip files will be almost indistinguishable from encrypted sectors by entropy. Another problem is that Shannon's entropy completely disregards the order of values. For example, simple counting up (x00 x01 ... xFE xFF) repeatedly, which is often part of files, results in the entropy of 1, despite this clearly not being random.

Most works I found that attempted to use entropy calculation to classify small data samples used Shannon's entropy despite its drawbacks mentioned above. However, each work aimed to use the calculated entropy differently. Some used[3] or tried to use[4] it to classify blocks for use in file carving and not encryption detection.

Other works used[6] or tried to use[7] entropy calculation as input or part of the input for machine learning trained to classify network packets. Work[6] also suggested using Tsallis entropy for calculation. However, the work did not attempt to calculate Tsallis entropy and instead decided to focus on Shannon's entropy.

Another work worthy of consideration[8] compared multiple entropy estimation algorithms. The work concluded by recommending the Miller-Madow method for uniform byte value distributions to estimate entropy. Entropy estimation will be helpful when considering the efficiency and speed of the entropy calculation.

Chi-squared test

The chi-squared test or χ^2 test is used to determine whether or not the data fit our expectations.[9] For example, consider flipping five fair coins and counting flipped heads. The probability distribution of the results (assuming that the coins cannot land on their side) would look like this:

number of heads	0	1	2	3	4	5
probability	$\frac{1}{32}$	$\frac{5}{32}$	$\frac{10}{32}$	$\frac{10}{32}$	$\frac{5}{32}$	$\frac{1}{32}$

First, we select a significance level (e.g. $\alpha = 0.05$). Then, after repeating the experiment of flipping five coins 160 times and adding up the results, we get the following table:

number of heads	0	1	2	3	4	5
number of flips (X)	2	8	34	64	44	8
expected number of flips (E)	5	25	50	50	25	5

Given counts of variables X_i, expected counts of variables $E_i = 160 *$ probability, and the number of columns n chi-square test statistic can be calculated using:

$$\chi^2 = \sum_{i=0}^{n-1} \left(\frac{(X_i - E_i)^2}{E_i} \right)$$

After getting the value approximation, the corresponding value from the cumulative chi-squared distribution for $n - 1$ degrees of freedom represents how likely was the measured data is from the distribution of our null hypothesis.

So, for the coin example, the chi-square statistic is calculated:

$$\chi^2 = \frac{(2-5)^2}{5} + \frac{(8-25)^2}{25} + \cdots + \frac{(8-5)^2}{5} = 38.64$$

After calculating the image of 38.64 under the chi-square cumulative distribution function for $n - 1$ degrees of freedom, we can see that $F_5(38.64) = 0.9999997$, which means that the p-value $= 1 - 0.9999997$ is smaller than our α, and we can therefore reject our null hypothesis, meaning that the fact that the in the measured data fair coins were used is less than 5%. And indeed, the obtained counts are from tests using two fair and three rigged coins with the probability of getting heads of $\frac{2}{3}$. After calculating the chi-square statistic for the hypothesis with rigged coins, we get $\chi^2 = 4.141$ and $F_5(4.141) = 0.4707$ and since $1 - 0.4707 > \alpha$, this hypothesis cannot be rejected.

As the chi-square test is only an approximation and gets more precise with more data, expected values should be at least 5, and it is preferable that they are much higher. [10]

For detection of random numbers, it is possible, for example, to create a column for each possible number, a column for ranges of numbers, or create a column for each remainder after division by a preselected number. Since the distribution of truly random numbers should be uniform, the expected value (E_i) should be the same for all columns. When the null hypothesis of uniformity of the numbers with sufficiently small α gets rejected, we can assume that the numbers are not random enough.

1.2 Visualization

After classifying all disk sectors based on byte patterns and entropy, it all comes down to visualizing the gathered data. While it would be certainly possible to draw a histogram of all sectors' entropy values or a pie graph based on detected patterns, this would not be as illustrative as the chosen approach, and much of the information about sector position in the disk would be lost. That is why the resulting utility visualizes the data using a bitmap, where each pixel represents a single sector on a disk.

Many works which were visualizing data used the most straightforward technique of *sweeping*. [11, 12, 13] This means that the first pixel is placed in the top-left corner, and each following pixel is placed to the right of the previous one except for when the position exceeds the fixed width of the image. In that case, the pixel is placed on the left-most position on the following line. This technique can be very illustrative in cases when the disk contains long sequences of equally classified sectors. However, when the disk would contain a shorter sequence, this would produce only a horizontal line with a single-pixel width, which could be hard to see and easily overlooked.

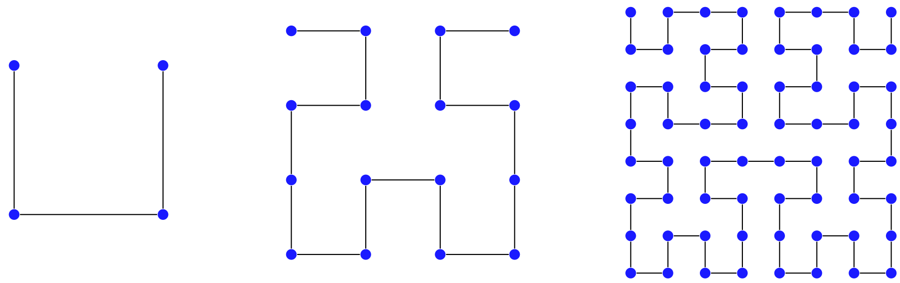

Figure 1.1: the first three Hilbert curve iterations

That is why the work [14] used the more complex Hilbert space-filling curve. The Hilbert curve passes through every pixel in a square exactly once in such a recursive pattern which always keeps consecutive pixels next to each other.[15] Moreover, placing pixels in these specific ways ensures that the shorter sequences are expanded into multiple lines and aggregated into clusters which makes them more easily visible. The curve covers a square with the side length of 2^i pixels (i.e., a total of 4^i pixels) where i is the number of iterations.

However, when using the Hilbert curve, another problem arises. There is no intuitive way to tell where the visualized sectors are located in the source image.

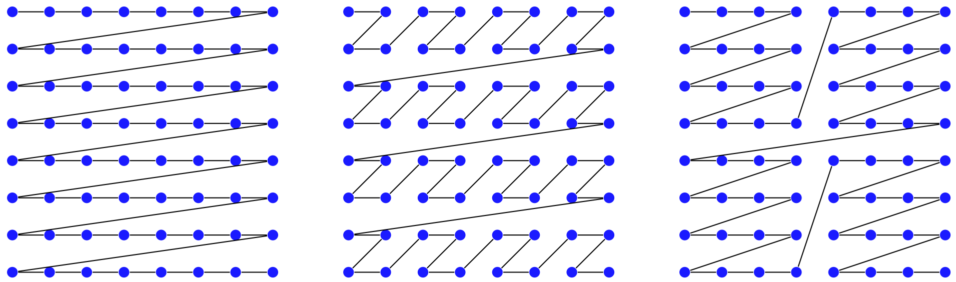

Figure 1.2: sweeping, 2x2 block sweeping, and 4x4 block sweeping

A middle ground between simple sweeping and the Hilbert curve would be block-sweeping. Block-sweeping uses the sweeping method to fill up a square NxN pixels in size, then continues to another NxN pixel block and places these pixel blocs in the same way simple sweeping would place individual pixels. This means that sweeping is just a version of block-sweeping, with pixel blocks of 1x1 pixels. By employing this technique, most shorter sequences are still more pronounced by getting expanded into multiple lines, and the position of pixels in the image more closely resembles the sector position in the source image than in the Hilbert curve. However, same as with sweeping, consecutive sectors are not always guaranteed to be right next to each other.

2 Used tools

2.1 Pillow

Pillow[16] is an image manipulation library for Python, which is a fork of the discontinued library PIL[17]. This library is used to visualize the analysis results. Since the results should be visualized as an image, where each pixel represents a single disk sector, statistical visualization libraries like Matplotlib[18], seaborn[19], or Gnuplot[20] were not good choices as they were not created with this exact type of visualization in mind. While they provide the means to create such visualizations, they are not as straightforward as the means provided by Pillow. While Pillow offers many more features beyond the very basics needed, it still keeps the interface for drawing one pixel at a time very simple.

2.1.1 Image

The module `PIL.Image` provides an essential toolkit for manipulating images. Given the image mode (e.g., RGB or RGBA) and image size, function `Image.new` creates an instance of the `PIL.Image.Image` class. The `Image` class stores the state of the resulting image and can be modified using its methods.

The method `Image.putpixel` modifies the state of the Image object and changes the color of the pixel on the given coordinates to the given color. `Image.save` tries to store the image on the provided path, and the method `Image.close` releases allocated memory. [21]

2.1.2 ImageDraw

The module `PIL.ImageDraw` is used to modify the `Image` class from `PIL.Image` in more powerful ways than just changing single pixels. Function `ImageDraw.Draw` creates a special context object for the given `Image` object, which can be used for further inplace modifications. The class of the context object `ImageDraw.ImageDraw` provides a wide range of shape drawing methods.

The method `ImageDraw.rectangle`, allows for drawing rectangle for provided coordinates and colors. It is also possible to specify the width and color of the rectangle outline. The method `ImageDraw.text` allows for writing a provided string in a font on the image. Both of these methods can be used for drawing the image legend. [22]

2.1.3 ImageFont

The module `ImageFont` is used with working with fonts with the Pillow library. It provides the means to load installed fonts by name or from path using the function `ImageFont.load` or to load a fallback font in case no other font is found with function `ImageFont.load_default`. Both of these functions return a `ImageFont.ImageFont` object, which contains a method `ImageFont.getsize` for calculating the dimensions in pixels of the box occupied by provided text written in this font. [23]

3 Implementation

4 Results

5 Conclusion

Bibliography

1. *Transition to advanced format 4K sector hard drives* [online] [visited on 2021-12-12]. Available from: https://www.seagate.com/tech-insights/advanced-format-4k-sector-hard-drives-master-ti/.
2. MCMILLEN, Wes. *White Paper: Western Digital Trim Command - General Benefits for Hard Disk Drives* [online]. 2021-12 [visited on 2022-03-05]. Tech. rep. Available from: https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/product/internal-drives/wd-purple-hdd/whitepaper-generic-benefit-for-hard-disk-drive.pdf.
3. FOSTER, Kristina. *Using distinct sectors in media sampling and full media analysis to detect presence of documents from a corpus*. 2012. Available also from: https://apps.dtic.mil/sti/citations/ADA570831. MA thesis. Naval Postgraduate School Montery CA.
4. GARFINKEL, Simson; MCCARRIN, Michael. Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb. *Digital Investigation*. 2015, vol. 14, S95–S105. Available from DOI: 10.1016/j.diin.2015.05.001.
5. SHANNON, C. E. A mathematical theory of communication. *The Bell System Technical Journal*. 1948, vol. 27, no. 3, pp. 379–423. Available from DOI: 10.1002/j.1538-7305.1948.tb01338.x.
6. WANG, Yipeng; ZHANG, Zhibin; GUO, Li; LI, Shuhao. Using Entropy to Classify Traffic More Deeply. In: *2011 IEEE Sixth International Conference on Networking, Architecture, and Storage*. 2011, pp. 45–52. Available from DOI: 10.1109/NAS.2011.18.
7. BEZAWADA, Bruhadeshwar; BACHANI, Maalvika; PETERSON, Jordan; SHIRAZI, Hossein; RAY, Indrakshi; RAY, Indrajit. Behavioral Fingerprinting of IoT Devices. In: *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security*. Toronto, Canada: Association for Computing Machinery, 2018, pp. 41–50. ASHES '18. ISBN 9781450359962. Available from DOI: 10.1145/3266444.3266452.

8. FIALLO, Ernesto; LEGÓN, C.M. Comparison of estimates of entropy in small sample sizes. 2019. Available from DOI: 10.13140/RG.2.2.19371.28960.

9. PEARSON, Karl. X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*. 1900, vol. 50, no. 302, pp. 157–175. Available from DOI: 10.1080/14786440009463897.

10. KNUTH, Donald Ervin. *The Art of Computer Programming Vol.2: Seminumerical Algorithms*. 2nd ed. Addison-Wesley, 1981. ISBN 0-201-03822-6.

11. HARGREAVES, Christopher. Visualisation of allocated and unallocated data blocks in digital forensics. In: 2013. Available also from: https://www.researchgate.net/publication/282538793_Hash-based_carving_Searching_media_for_complete_files_and_file_fragments_with_sector_hashing_and_hashdb.

12. CHARALAMPIDIS, Ioannis. *Visualising Filesystems* [online]. 2018-11-08 [visited on 2021-12-14]. Available from: https://www.linkedin.com/pulse/visualising-filesystems-ioannis-charalampidis.

13. BROŽ, Milan. *TRIM & dm-crypt ... problems?* [Online]. 2011-08-14 [visited on 2021-12-14]. Available from: https://asalor.blogspot.com/2011/08/trim-dm-crypt-problems.html.

14. CORTESI, Aldo. *Visualizing binaries with space-filling curves* [online]. 2011-12-23 [visited on 2021-12-14]. Available from: https://corte.si/posts/visualisation/binvis/.

15. HILBERT, David. Ueber die stetige Abbildung einer Linie auf ein Flächenstück. 1891. Available also from: http://www.digizeitschriften.de/en/dms/img/?PID=GDZPPN002253127&physid=phys476#navi.

16. KEMENADE, Hugo van; MURRAY, Andrew; WIREDFOOL; CLARK, Alex; KARPINSKY, Alexander; BARANOVIČ, Ondrej; GOHLKE, Christoph; DUFRESNE, Jon; SCHMIDT, David; KOPACHEV, Konstantin; HOUGHTON, Alastair; MANI, Sandro; LANDEY, Steve; VASHEK; WARE, Josh; DOUGLAS, Jason; T., Stanislau; CARO, David; MARTINEZ, Uriel; KOSSOUHO, Steve; LAHD, Riley; LEE, Antony; BROWN, Eric W.; TONNHOFER, Oliver; BONFILL, Mickael; ROWLANDS, Peter; AL-SAIDI, Fahad; NOVIKOV, German; GÓRNY, Michał. python-pillow/Pillow: 9.0.1. 2022. Available from DOI: `10.5281/zenodo.5953590`.

17. *Python Imaging Library* [online] [visited on 2022-02-28]. Available from: `https://web.archive.org/web/20150316203935/http://effbot.org/zone/pil-index.htm`.

18. HUNTER, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. 2007, vol. 9, no. 3, pp. 90–95. Available from DOI: `10.1109/MCSE.2007.55`.

19. WASKOM, Michael L. seaborn: statistical data visualization. *Journal of Open Source Software*. 2021, vol. 6, no. 60, p. 3021. Available from DOI: `10.21105/joss.03021`.

20. Gnuplot [online]. [N.d.] [visited on 2022-02-11]. Available from: `http://www.gnuplot.info/`.

21. *Image Module* [online]. [N.d.] [visited on 2022-03-14]. Available from: `https://pillow.readthedocs.io/en/stable/reference/Image.html`.

22. *ImageDraw Module* [online]. [N.d.] [visited on 2022-03-14]. Available from: `https://pillow.readthedocs.io/en/stable/reference/ImageDraw.html`.

23. *ImageFont Module* [online]. [N.d.] [visited on 2022-03-14]. Available from: `https://pillow.readthedocs.io/en/stable/reference/ImageFont.html`.