MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Disk sector content analysis and visualization

Bachelor's Thesis

JAKUB MALOŠTÍK

Brno, Fall 2021

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, Fall 2021

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

1.1 Pattern detection

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 0x00 often signalizes that this sector has not been used or that the file stored there has been shredded, or bytes 0x55 0xAA at the end of the sector would signalize a master boot record (MBR). However, a closer inspection is required because checking for only two bytes would cause way too many false positives in encrypted sectors.

Most works focusing on detecting patterns of bytes on sectors [2] [3] 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. [2]

1.2 Calculation of entropy

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 for the predictability of the byte values by calculating byte entropy in order to estimate whether or not the sector is encrypted. 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:

$$\mu(S) = \frac{H}{H_{max}} = \frac{-\sum_{i=0}^{255}(P(x_i)\log_2(P(x_i)))}{-\sum_{i=0}^{255}(\frac{\frac{s}{256}}{s}\log_2(\frac{\frac{s}{256}}{s}))} =$$

$$= -\frac{1}{8}\sum_{i=0}^{255}(P(x_i)\log_2(P(x_i)))$$

Where s is equal to the sector size in bytes. Normalized Shannon's entropy ranges from 0, the least random, which means it contains a single repeated byte value, to 1, the most random, which means that every byte value is contained in the sector an equal amount of times (i.e., exactly $\frac{s}{256}$ times). Using this value, one can estimate whether the sector contains encrypted data.

However, multiple problems arise when using Shannon's entropy. A single sector of data is not enough to calculate entropy reliably. In order to get precise results, the provided data would need to be in the order of gigabytes. Moreover, 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 (0x00 0x01 ... 0xFE 0xFF) 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 [2] or tried to use [3] it to classify blocks for use in file carving and not encryption detection.

Other works used [4] or tried to use [5] entropy calculation as input or part of the input for machine learning trained to classify network packets. Work [4] 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 [6] 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.

1.3 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 *scanning*. [7, 8, 9] 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. That is why work [10] used other, more complex, space-filling curves like the Z-order curve (also known as Morton curve) or Hilbert curve. Placing pixels in these specific ways ensures that the shorter sequences are expanded into multiple lines and become easily visible while keeping consecutive sectors close together.

2 Used tools

2.1 Python

I chose Python 3[11] based on several factors. The most significant factor was subjective – my comfortability with the language. Based on this, my decision was mainly between C and Python.

Another factor is efficiency. While this utility is not time-sensitive and has no real-time constraints, its efficiency is very desirable and can be the difference between analysis taking minutes and taking hours or even days. Here, C is the obvious winner. However, Python enables C integration[12], which could be used to implement parts of code called tens of millions of times to speed up the utility's performance.

Another consideration is the extensibility of the written code. One of my goals is to make the code as extensible as possible. While it is certainly possible to make easily extensible code in C, I did not feel confident enough to do it well.

The next significant consideration was the availability of other tools and built-in libraries. Python provides a wide range of built-in modules, making programming easier like argparse[13] and collections[14] while also having many other installable libraries, which are mostly platform-independent.

2.2 Pillow

Pillow[15] is an image manipulation library for Python, which is a fork of the discontinued library PIL[16].

I am using this library to visualize the analysis results. Since I decided on the approach of visualizing the results as an image, where each pixel represents a single disk sector, statistical visualization libraries like Matplotlib[17], seaborn[18], or Gnuplot[19] were not good choices as they were not created with this exact type of visualization in mind and 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.

2.3 Scipy

Scipy[20] is a Python library for scientific computing. I am using this library to calculate the boundaries of the chi-square statistic of random data with inverse cumulative distribution function for the chi-square distribution based on given degrees of freedom (formula) and provided p-value boundaries. I am also using this library to calculate the Kolmogorov–Smirnov test.

3 Entropy calculation

4 Ways of visualization

5 Implementation

6 Results

7 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. 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.
3. 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`.
4. 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`.
5. 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`.
6. 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`.
7. 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`.

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

9. 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`.

10. 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/`.

11. *Python* [online]. [N.d.] [visited on 2022-02-11]. Available from: `https://www.python.org/`.

12. *Extending Python with C or C++* [online] [visited on 2022-02-11]. Available from: `https://docs.python.org/3/extending/extending.html`.

13. *argparse – Parser for command-line options, arguments and sub-commands* [online] [visited on 2022-02-14]. Available from: `https://docs.python.org/3/library/argparse.html`.

14. *collections – Container datatypes* [online] [visited on 2022-02-14]. Available from: `https://docs.python.org/3/library/collections.html`.

15. 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*. Zenodo, 2022. Version 9.0.1. Available from DOI: `10.5281/zenodo.5953590`.

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

17. 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`.

18. 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`.

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

20. VIRTANEN, Pauli; GOMMERS, Ralf; OLIPHANT, Travis E.; HABERLAND, Matt; REDDY, Tyler; COURNAPEAU, David; BUROVSKI, Evgeni; PETERSON, Pearu; WECKESSER, Warren; BRIGHT, Jonathan; VAN DER WALT, Stéfan J.; BRETT, Matthew; WILSON, Joshua; MILLMAN, K. Jarrod; MAYOROV, Nikolay; NELSON, Andrew R. J.; JONES, Eric; KERN, Robert; LARSON, Eric; CAREY, C J; POLAT, İlhan; FENG, Yu; MOORE, Eric W.; VANDERPLAS, Jake; LAXALDE, Denis; PERKTOLD, Josef; CIMRMAN, Robert; HENRIKSEN, Ian; QUINTERO, E. A.; HARRIS, Charles R.; ARCHIBALD, Anne M.; RIBEIRO, Antônio H.; PEDREGOSA, Fabian; VAN MULBREGT, Paul; SCIPY 1.0 CONTRIBUTORS. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*. 2020, vol. 17, pp. 261–272. Available from DOI: `10.1038/s41592-019-0686-2`.