Visualizing filesystem metadata using Highcharts

IMT4641 Computational Forensics
ANDRÉ NORDBØ
Gjøvik University College (12HMISA)

Abstract

In this paper we will look at how filesystem metadata can be explored using interactive visualization techniques. By taking the output of Sleuthkit's FLS command, a web based tool built in python using existing javascript modules has been asembled enabling an analyst to explore the metadata at different scales.

1. INTRODUCTION

This project is the practical part continuing from my digital forensics II project[Nordbø, 2013] where the main focus was on highlighting important aspects of visualizing, challenges in visualization and examples of visualization usage in the forensics field. In this project a prototype of a file system meta data visualizer has been created, inspired by the Sleuthkit and Autopsy.

A filesystem can contain several hundred thousands of files, and the files can be user generated like documents and pictures, or it can be program and system files. Figuring out where to look for evidence in such large amounts of data is very tedious and relies on expert knowledge. In the early days of personal computers malware often dumped executables directly on the root of the C:/ drive, and anyone who dared opening "My computer" would be able to spot the abnormal files, although only experienced users would react on the findings. During labs in digital forensics II we mounted and looked for evidence on images. Getting an overview of the files was difficult and is the main motivation for creating this tool so to help in determining what to focus on.

In previous work in the field we know that Sleuthkit has a graphical user interface called Autopsy (as in post-mortem examination), and in version 2 it was a web based framework running a local webserver. Version 3 now in beta has moved to a stand alone application and it has "timeline analysis"[1] mentioned as a feature. During development of this application this tool was tested out. The graphing function did not work, but this blog[Letourneau, 2013] explains what it should have looked like. Another tool mentioned at the Sleuthkit webpage is Zeitline[2].

[1] See description on `http://www.sleuthkit.org/autopsy/timeline.php`
[2] `http://www.dfrws.org/2005/proceedings/buchholz_zeitline.pdf`

2. MAIN CONTENT

2.1 Criteria for design

Selection of platform: When writing a desktop application, one of the major concerns are compatibility across operating systems and architectures. A huge trend the last couple of years is to move applications to the web using the browser as the main portal and web standards such as HTML, CSS and JavaScript are available on most systems. Good examples of this trend is Google docs allowing simultaneous cooperation on documents in real time. There is a cost penalty in terms of speed when using such high level programing languages and one of the biggest questions is how to distribute the workload between the server and the client browser, especially when we know the processing power of small devices are orders of magnitude weaker than high end desktops. Another major concern is control of the content being stored in a separate location, not on the desktop computer. A solution is to run the services on the local network instead of outsourcing it like in the case with Google.

Server side technology: There are many web application solutions for all the major programing languages, including ASP, C++, Java (which must not be confused with JavaScript), Perl, PHP, Python and Ruby. When implementing a prototype, both development speed and iteration are important aspects, and a known language will speed up the process, therefore Python using the framework Django was chosen.

For graphing there are two distinct directions to take: Render graphs on the server and send them as raster images or vector images to the client, or use client side technology to render it on the fly using the raw data. For drawing graphs client side there are also many frameworks to chose among: Flot, Chart.js, Highcharts, gRaphaël, CanvasJS, d3.js to mention some of them[3]. Highcharts[Highsoft Solutions AS,] was chosen for this prototype.

[3] More examples at `http://sixrevisions.com/javascript/20-fresh-javascript-data-visualization-libraries`

2.2 Description of the application

The general framework is shown in this illustration (figure 1), and shows the interaction between users and the server.

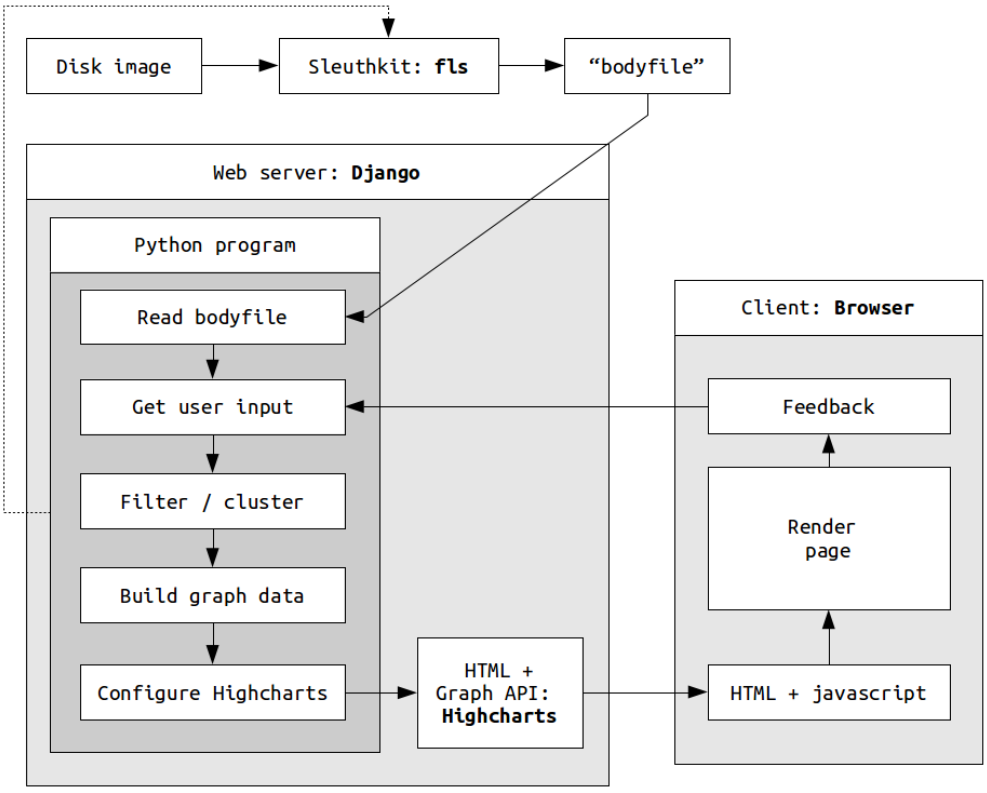

Figure 1: *How the application is functioning*

As with all web based application, the user will ask for a page, the server will generate the page, send it to the user, the user will interact with it, resulting in a new query for the server. The main input is a body file generated by SleuthKit fls command. The output is often sent to another tool called mactime in order to sort it, but this step is skipped because mactime will split up individual file meta data for each unique timestamp. The output of fls will be determined by the filesystem of the image analyzed. This is how it looks like for Linux EXT3:

```
0|/etc/passwd|22987|r/rrw-r--r--|0|0|1273|1247426478|1244827526|1244827526|0
file type | full path | inode | permissions | owner | group | size | accessed | modified | created | 0
```

And this is how it looks like for Windows NTFS:

```
0|C:/Windows/explorer.exe|50563-128-4|r/rrwxrwxrwx|0|0|2614784|1313238285|1298698387|1313357703|1313238285
file type | full path | metadata address | permissions | 0 | 0 | size | accessed | modified | created | birth
```

The program, located in "webvisu/filemeta/views.py" will load and store the content of these files into a list in memory, based on user input it will filter out lines, and build series for display in highcharts. Currently these filters are implemented:

- Path include and exclude using OR logic between words separated by space

- Datetime start and end filtering
- Date type: created, accessed and modified (birth could be added, but does not do anything on linux file systems)

Because of the huge amounts of data, the program will switch between showing files and histogram based on a variable "GRAPH_HARD_LIMIT" set at 1500 data points, measured after filtering. In histogram mode (figure 2) the user can select an area, and the page will reload to show only that region in time, based on the selected date category. When the amount of data is below the limit, a bubble graph will show up placing every file in a coordinate system with time on the x-axis, size on the y-axis and the radius according to the inode as shown in figure 5. (figure 4 has inode on the y-axis and size on the radius). Including and excluding parts of the path is another way to limit the amount of points as shown in figure 3.

One interesting aspect of plotting actions based on time, is that it's very easy to see that some files, typically when installing software will have timestamps very close in time, and when looking at human generated content it will be spread out in a more random pattern as discussed in digital forensics I. Some timestamp modifications might also be possible to spot based on different inode number, although one must be very careful when interpreting such information since inodes are being reused. It's also important to be aware that the content of compressed folders at this level is not visible

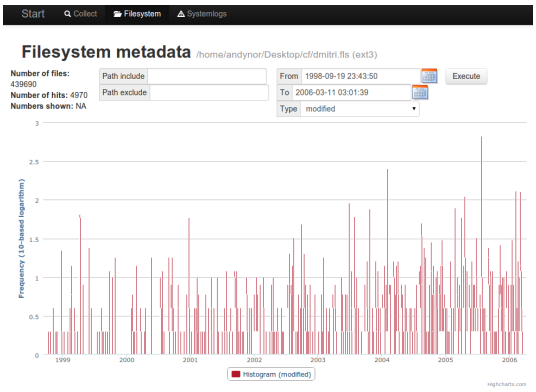

Figure 2: *Histogram when > limit*

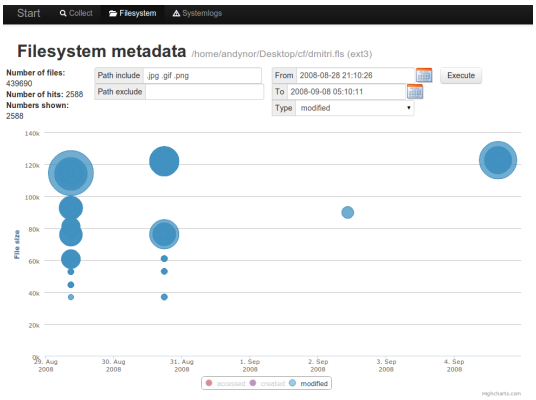

Figure 3: *Image file ending search*

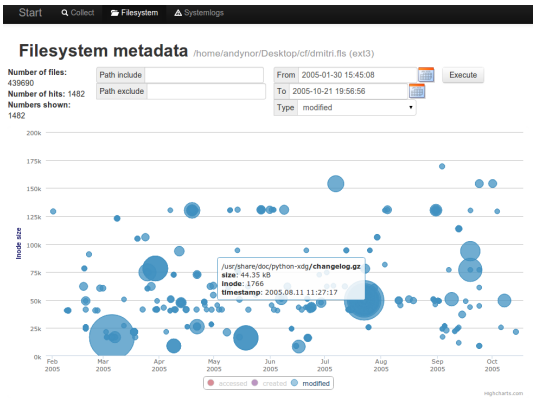

Figure 4: *Individual files: y=inode*

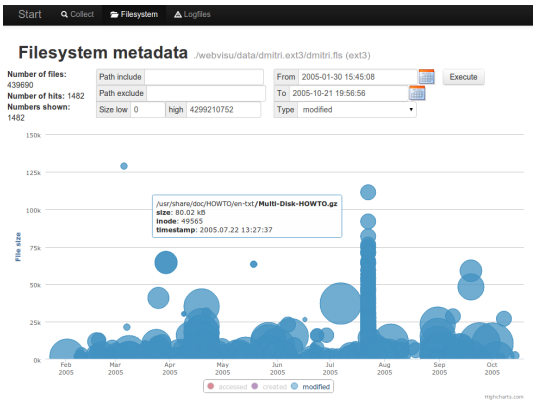

Figure 5: *Individual files: y=size*

2.3 Test data

Two images "dmitri.dd" (ext3) and "jotunheim.img" (ntfs) were used for testing. They were both used in previous labs during digital forensics I and II. The fls output files are included in the source package, selectable from a menu.

2.4 Deployment

First some prerequisites must be in place: Python (v2.7) and Django (v1.4.2) must be present for serving the application. They are available on many platforms including Windows and Linux. For Ubuntu:

```
sudo apt-get install python-pip
sudo pip install django
```

For testing purposes, navigate to the root of the django app and find the manage.py script. In order to start the test server built in to Django (not for production!), run

```
python manage.py runserver 0.0.0.0:8000
```

Visit this URL (localhost:8000) using a modern browser. Google Chrome is recommended because of it's javascript performance. In order to generate the body file you need to install sleuth kit and run the command

```
fls -l -r -m *mount-point* -o *image-offset* -z UTC *image* » *output-file*
```

on the acquired image. Descriptions of this process is included inside the web interface. Highcharts is included in the package and is licensed under as creative commons attribution-noncommercial meaning it cannot be used in a corporate environment without a developer license.

When deploying a django application for actual usage, best practices is to add it as a module to a real webserver. One easy combination is using Apache + mod_wsgi[Django documentation,]. Also remember to disable debug mode in "settings.py". This prototype has few input validation mechanisms in place.

2.5 Performance

On the server side there are two concerns: cpu usage (time) and memory footprint. Time is measured before loading the fls file into memory and right before the content is sent to the user. Typical timings for the full dmitri image with 439 690 files was about 2 - 2.5 second per request, depending on filters, and closer to 3 seconds after adding file size filtering. (Using single core Intel I5 3570K)

Memory footprint is about 270MB, still using the dmitri image being 56MB, and a simple test with the double amount of data (file appended to itself) gives a footprint of 537MB. If this trend is consistent, the required memory is about 4.8x the size of the fls output file. It must be noted the python program is storing strings directly in a list (array) which has overhead, and while building the chart series graphs, both the source and the new variable must be kept in memory.

The current implementation calculates everything from scratch for every request. Buffering of intermediate results is a common way to speed up performance, and especially clustering as mentioned in the **improvement** section is a good candidate.

On the client side the main two concerns are load time of the html document and the performance of the graphing software. The load time consists of server generation time + transfer time. Transfer time depends on bandwidth and the size of the graph data. Compression [4] server side should be enabled.

[4] One example for Apache is mod_deflate

Graph data is restricted by the "GRAPH_HARD_LIMIT" variable and at the same time it will restrict the amount of work required by the browser rending the graph points.

3. IMPROVEMENTS

When working on a problem, many thoughts on improvements and features are uncovered.

- Toggling between inode and size on the y-axis
- Deal with lots of data by clustering. The main idea is that by observing a lot of files are grouped very close when visualized, more data points could have been squeezed in if a noisy event were replaced by an aggregate. Clustering only based on time interval would be too naive, but if path information were included, and perhaps also "close" inode number, then an installation and an unrelated file could be visible at the same time.
- Normalize the inode number in a fixed range depending on view (i.e. between 1-10)
- Add integrity hash, fuzzy hash and filetype based on magic number to each file
- Use integrity hash for filtering known good and bad
- Use fuzzy hash to display similar files and find similar files based on example
- Possibility to choose between AND / OR logic in filters (Or between key words implemented and AND between fields)
- Automate the process of generating the fls output (upload image and point to it)
- Selectable files to selection list: Export actual files using icat
- Improve speed: usage of databases? Parallel execution in a computer farm (integrate with Hadoop?)
- Extend the framework to include a module for log2timeline: A tool for extracting and sorting the content of known log files.

4. CONCLUSION

Filsystems contains huge amounts of information, and being able to present it in an intuitive way for human investigators is important. Computational forensics is all about using computers to automate and make sense of digital data with all the benefits of computers doing the tedious error prone work and the human doing the creativity and hypothesis generation. In this paper a simple web based timeline graphing application has been prototyped and it's benefits and weaknesses has been discussed. The main idea is to use bubble charts to show timing, size and file id (inode) in the same view. Weaknesses with this method is low performance and high memory usage.

REFERENCES

[Django documentation,] Django documentation. How to deploy with wsgi. `https://docs.djangoproject.com/en/dev/howto/deployment/wsgi/`. Visited June 2013.

[Highsoft Solutions AS,] Highsoft Solutions AS. Highcharts js - interactive javascript charts for your web projects. `http://www.highcharts.com/products/highcharts`. Visited June 2013.

[Letourneau, 2013] Letourneau, J. (2013). Autopsy feature: Graphical timeline analysis for cyber forensics (posted 23rd may 2013). `http://info.basistech.com/blog/?Tag=sleuthkit.org`. Visited June 2013.

[Nordbø, 2013] Nordbø, A. (2013). Data visualization for discovery, analysis and presentation of digital evidence. Project report in digital forensics II.