



ICT2202 Assignment One Report

Team Members	E-Mail
SEET JU LENG ABNER MANUEL	1801329@sit.singaporetech.edu.sg
TAN WEI QI	1800860@sit.singaporetech.edu.sg
TAN KAH WEI	1800769@sit.singaporetech.edu.sg
DEEPAK JOSHI	1800546@sit.singaporetech.edu.sg
JONATHAN YEO SHI HAO	1800553@sit.singaporetech.edu.sg

Table of contents

- 1. Background Research**
- 2. Issues and Challenges**
- 3. Existing Solutions**
- 4. Our Solution (BDE_Analyser)**
- 5. Dependencies**
- 6. Architecture and Engineering**
- 7. Test setup and results**

1. Background Research

Most modern filesystems rely on directories to map filenames to inode numbers. What happens if the directory data gets corrupted? Of course, there are bulky file recovery tools that carve files without the filesystem metadata. However, those available in the internet lack in many areas, mainly:

- does not provide simple and/or detailed analysis before jumping to carving
- intensive in resource, storage, and library
- unable to dynamically calculate accuracy for each action
- does not let the user pick from multiple possible outcomes for the same operation
- proprietary, payware

This tool (BDE_Analyser) will be a great help to forensic investigators as they can use a tool that addresses all the above-mentioned issues encountered when trying to do forensics on binary data. The features of BDE_Analyser (as of v1.0) include:

- provides simple analysis (Level 1 test) and detailed analysis (Level 2 test), before carving (Level 3)
- it's a lightweight and portable python file, with only built-in libraries and no external library
- dynamically calculates accuracy for each action, so user knows how reliable it is
- gives user multiple outcomes to choose from for carving operations
- open source, freeware

2. Issues and Challenges

Issues and challenges with solving the problem includes the fact that the file headers in magic numbers depend on a value called “offset”. Without a valid judgement of which is the starting byte, it would be hard to match the signature with the database, and hence might result in inaccurate prediction.

Also, using Machine Learning to detect filetype with Artificial Intelligence studying the binary data is an interesting idea, however, in the option we explored, the machine was not able to learn much dynamically, at least not enough to provide results with significant accuracy. Since, accuracy was of the essence, we had to expand out options, which led us to the BDE_Analyser.

3. Existing Solutions

One of the most readily available existing solution to this problem is the Linux’s “file” command. It is available as a port to windows, <http://gnuwin32.sourceforge.net/packages/file.htm>, which we used as a reference for coding our tool. We did not copy any codes, but rather used it as a baseline to detect if our program surpasses their program, in terms of detections and accuracy.

The good thing about the Linux’s “file” command is that it is preinstalled in most linux distributions, so forensic investigators automatically have access to it. Which is possibly the only advantage it has over our program, which has not published to Linux repositories yet.

Other carving and file recovery solutions are not mentioned because they do not have multiple analysis stages; they’re heavy on resource, storage, libraries; does not dynamically calculate accuracy for every action; does not let the user pick from the list of outcomes; and most importantly may not be open source.

4. Our Solution (BDE_Analyser)

https://github.com/1800546/BDE_Analyser

We named our tool Binary Data Extract Analyser, or BDE_Analyser for short.

This tool is designed to analyse a chunk of raw binary data extracted from filesystem, and identify the filetype of unfragmented file(s) present inside, and even carve it/them out if supported filetype(s).

The coding did not use any OS-specific libraries or features, so it should most likely be cross-platform, but that has not been tested yet. So, for now, the recommended system configuration would be Windows 10 x64, Python 3.7.2, and Intel i5(5th Gen) or better.

The program runs on python console, and is interactive. We didn't include a silent/background mode since our program relies on user input to make decisions on the choice of algorithms used. If it runs in silent mode, meaning the highest level (Level 3) of intensity will have to be used to ensure suitability for all scenarios, and that would be resource wasting for scenarios without the need for high-intensity algorithms. So, if the user has python installed on their Windows 10 x64 machine, they can just run the tool natively from their Windows Powershell, provided they have added the python to the Windows's path environment variable.

To understanding the terminology used in the program, it is necessary to get the concept of the 3 Levels of progressively increasing testing algorithms. The Level 1 tests are very similar to the Linux's file command. Might not work for most complex cases, because it uses the "offset" of Magic Numbers, which is ineffective (as explained in the "Issues and Challenges" section above). The Level 2 tests are an in-depth scanning of

chunks. This scan can detect way more than the previous level, and is not limited to the “offset” that holds back the Level 1 tests. Hence, this layer test is particularly useful to determine the start point of file(s) in the chunk, not to mention it also provides the probability that the file exists and the probability that the file is of the mentioned filetype, even before proceeding to Level 3 tests. To deal with the end-point, we have Level 3 tests, that not only determine the end-point of a file, but also will carve it out for you, provided it is one of the supported filetypes (gif, png). Only limited filetypes are eligible for Layer 3 tests (as denoted by Layer 2 Special Pass, compared to ineligible Layer 2 Normal Pass) as of BDE_Analyser v1.0, because this is a proof-of-concept release, but in future versions support for more filetypes will be added.

The levels described above are tabulated here for better understanding:

Result	Eligibility		
	Lvl 1	Lvl 2	Lvl 3
Lvl 1 Fail	Completed	Yes	Possible (Lvl 2 Special Pass)
Lvl 1 Pass	Completed	Yes	Possible (Lvl 2 Special Pass)
Lvl 2 Fail	Completed	Completed	No
Lvl 2 Normal Pass	Completed	Completed	No
Lvl 2 Special Pass	Completed	Completed	Yes
Lvl 3 Fail	Completed	Completed	Completed
Lvl 3 Pass	Completed	Completed	Completed

In order to get best results, and for the convenience of the user, there are some recommendations when importing a chunk from a forensic disk image. If the user suspects that there is/are file(s) from sector x to sector y of an image, he/she can cut out the sector w (conservative start) to sector z (conservative finish). Although cutting x to y might work, it may not always be the best idea especially if the user cant identify the filetype. Hence, to avoid any harm to the files, the user can cut conservatively and let this tool do the rest of the work. It is up to the user on how conservative they would like to be. To be safe, if it can be afforded, the tips of the file (especially the ending) can be as prolonged as possible, to increase the probability of having an intact file in the chunk for the tool to recover.

As for the limitations of the tool, for a level 2 scan, the probability % is assuming the input is a single-file chunk, and the probability % will not be applicable to a multi-file

chunk as there is no algorithm in the tool for multi-file chunk probability % at level 2 scan stage. Here is a table describing the probability metrics in detail:

Result	Accuracy of Result	
	Single-file chunk	Multi-file chunk
Lvl 1 Fail	50%	No algorithm
Lvl 1 Pass	100%*	No algorithm
Lvl 2 Fail	100%*	100%*
Lvl 2 Normal Pass	Calculated during runtime	No algorithm
Lvl 2 Special Pass	Calculated during runtime	No algorithm
Lvl 3 Fail	100%*	100%*
Lvl 3 Pass	100%	100%

*These values assume that the filesystem metadata got corrupted. If it was intentionally tampered with, the values would be different.

5. Dependencies

This tool uses 5 of the python3's native libraries, which are os, json, glob, binascii, and shutil. Since those come with the python3 installation, we don't have to worry about that.

The only thing that needs to be noted is that in order to start the tool, the user can just type "python .\BDE_Analyser.py" in Windows Powershell once navigated to the project directory. Starting it from other directories using the full path might not work. Also, all input files for the project has to be placed in the project folder.

Another core feature the tool relies on is the "data.json" file in the project folder. It is an updated list of file signatures, and mainly used for Level 1 tests meant to emulate the Linux "file" command using the magic numbers. Currently, it uses the python3's native json library. There is an option to use external libraries such as simplejson (<https://pypi.org/project/simplejson/>), that uses C extensions rather than relying on pure python. The C extensions would make the tool run much

faster, but we decided against it as we felt that that the additional speed is negligible in our case due to the json dataset being little. Also, if we look at the testing speeds for all the tests offered by the our tool, it's not the Level 1 tests that are slow.

Test	Speed
Lvl 1	Instantaneous
Lvl 2	Proportional to chunksize
Lvl 3	Proportional to chunksize

During Level 2 tests, there is a “hexdump” file created in the project folder. This file will be used as a reference throughout the results of the Level 2 tests, which will mention the positions relative to the hexdump where certain files are detected. The files recovered using Level 3 tests are placed in the “recovered” folder.

6.Architecture and Engineering

As mentioned in the sections above, this code is designed for chunks with single files, although it supports chunks with multiple files too. However, the algorithm to calculate accuracy % would be off if multiple files are present. This is because the chunks are supposed to be cut conservatively before the startpoint and conservatively after the endpoint. Here are the recognition's success and failure parameters for both single-file and multi-chunk files:

Result	Single-File recognition	
	Linux file command	BDE_Analyser
Lvl 1 Fail	No	Possible (Lvl 2 Pass)
Lvl 1 Pass	Yes	Yes
Lvl 2 Fail	No	No
Lvl 2 Normal Pass	Possible (Lvl 1 Pass)	Yes
Lvl 2 Special Pass	Possible (Lvl 1 Pass)	Yes
Lvl 3 Fail	No	No
Lvl 3 Pass	Possible (Lvl 2 Spl Pass)	Yes

Result	Multi-File recognition	
	Linux file command	BDE_Analyser
Lvl 1 Fail	No	Possible (Lvl 2 Pass)
Lvl 1 Pass	No	Possible (Lvl 2 Pass)
Lvl 2 Fail	No	No
Lvl 2 Normal Pass	No	Yes
Lvl 2 Special Pass	No	Yes
Lvl 3 Fail	No	Possible (Non-“special” files)
Lvl 3 Pass	No	Yes

Also, now that we know under which circumstances can we get recognition using the different tools, lets take a look at the circumstances for which we have valid files present, for both single-file and multi-file chunks:

Result	Valid Single-File Presence	
	Linux file command	BDE_Analyser
Lvl 1 Fail	No	No
Lvl 1 Pass	Yes (Source)	Yes (Source)
Lvl 2 Fail	No	No
Lvl 2 Normal Pass	No	Possible (Level 1 Pass)
Lvl 2 Special Pass	No	Possible (Level 1/3 Pass)
Lvl 3 Fail	No	No
Lvl 3 Pass	No	Yes (Recovery)

Result	Valid Multi-File Presence	
	Linux file command	BDE_Analyser
Lvl 1 Fail	No	Possible (Level 3 Pass)
Lvl 1 Pass	No	Possible (Level 3 Pass)
Lvl 2 Fail	No	No
Lvl 2 Normal Pass	No	No
Lvl 2 Special Pass	No	Possible (Level 3 Pass)
Lvl 3 Fail	No	No
Lvl 3 Pass	No	Yes (Recovery)

7. Test setup and results

There are 15 sample input in the project folder to try with BDE_Analyser – namely testfile01 to testfile14, and cutoffdemo. This will give a rough idea of how it works.

The results of Level 1 test is displayed on the console's standard output, and the Level 2 results are a mixture of the console's standard output and the hexdump file in the project folder. Finally, for Level 3 results, they will be in the “recovered” folder in the project directory.

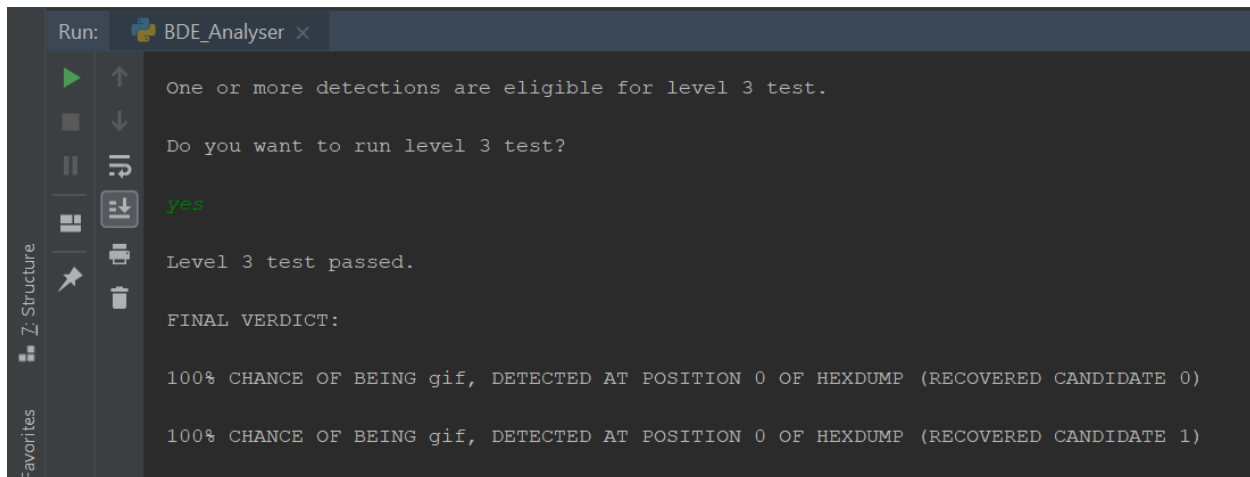
Firstly, lets look at the testfiles01 to testfiles14:

Result		Sample file	Display message	
			Linux file command	BDE_Analyser
Single-file chunk	Lvl 1 Fail	testfile01	data	UNKNOWN AS OF LEVEL 1 TEST
	Lvl 1 Pass	testfile02	GIF image	VARYING CHANCE OF BEING: ['gif'], DETECTED AT POSITION 0 OF CHUNK
	Lvl 2 Fail	testfile03	data	CHUNK HAS NO VALID FILE(S)
	Lvl 2 Normal Pass	testfile04	data	32% CHANCE OF BEING bmp, DETECTED AT POSITION 216 OF HEXDUMP
	Lvl 2 Special Pass	testfile05	data	52% CHANCE OF BEING png, DETECTED AT POSITION 216 OF HEXDUMP
	Lvl 3 Fail	testfile06	data	CHUNK HAS NO VALID FILE(S)
	Lvl 3 Pass	testfile07	data	100% CHANCE OF BEING gif, DETECTED AT POSITION 160 OF HEXDUMP (RECOVERED CANDIDATE 0)
Multi-file chunk	Lvl 1 Fail	testfile08	data	UNKNOWN AS OF LEVEL 1 TEST
	Lvl 1 Pass	testfile09	PNG image	Warning: Level 1 test does not check for multiple files in the chunk
	Lvl 2 Fail	testfile10	data	CHUNK HAS NO VALID FILE(S)
	Lvl 2 Normal Pass	testfile11	data	Level 2 test encountered the possibilities: ['ico', 'ttf', 'bmp']
	Lvl 2 Special Pass	testfile12	data	Level 2 test encountered the possibilities: ['png', 'gif', 'cat', 'tar.z', 'aac']
	Lvl 3 Fail	testfile13	data	23% CHANCE OF BEING bmp, DETECTED AT POSITION 324 OF HEXDUMP
	Lvl 3 Pass	testfile14	data	Level 3 test passed. (RECOVERED CANDIDATE 0) (RECOVERED CANDIDATE 1)

Now, let's take a look at the cutoffdemo file, which is a testfile meant to demonstrate a different kind of results, which the console output might not be able to show. This is to showcase how the Level 3 Pass can generate multiple outcomes, and let the user choose which would be the most appropriate. This would be most common for gif files, and less common for other files, due to the gif format being more susceptible to erroneous cutting.

The cutoffdemo chunk has 1 gif file inside without the filesystem metadata.

If the cutoffdemo were to undergo the Level 3 test, this would be the console output:

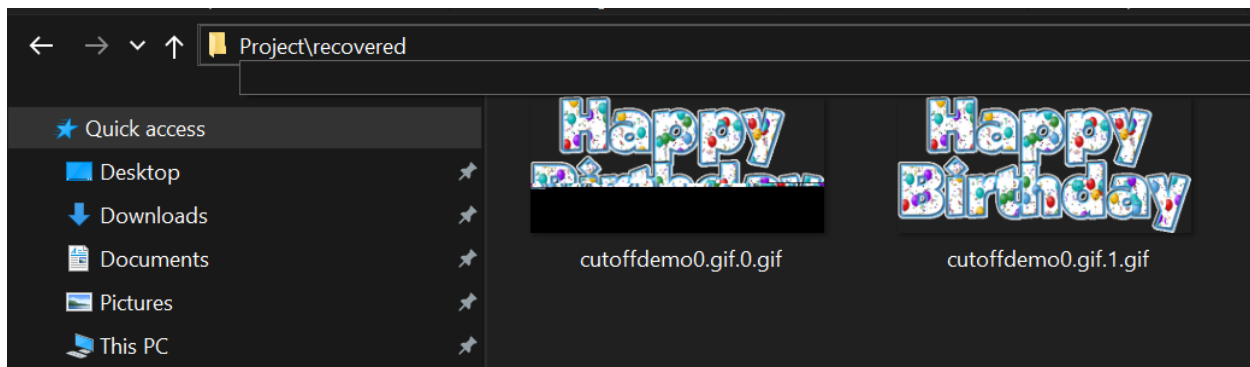


```

Run: BDE_Analyser x
One or more detections are eligible for level 3 test.
Do you want to run level 3 test?
yes
Level 3 test passed.
FINAL VERDICT:
100% CHANCE OF BEING gif, DETECTED AT POSITION 0 OF HEXDUMP (RECOVERED CANDIDATE 0)
100% CHANCE OF BEING gif, DETECTED AT POSITION 0 OF HEXDUMP (RECOVERED CANDIDATE 1)
  
```

However, that's not the interesting part, as the interesting part is when we further explore the 2 recovered candidates that was offered.

Now, let's take a look at the "recovered" folder:



If you notice, the candidate 0 file was abruptly cropped off whereas the candidate 1 file is the full canvas. That is because the BDE_Analyser sweeps the entire chunk using a special algorithm based on heuristics to discover multiple endpoints, and for every possible endpoint it detects, it carves a file. Hence, we have multiple files as a result for the Level 3 test of cutoffdemo.

Now, the user can choose which files he/she would like to keep. Not many tools in the market offer this feature. They either use the too conservative algorithm and always carve the file with overhead, or use erroneous algorithm that carves early resulting in an abruptly-cut-off file. That is a dangerous error as not all file formats are tolerant of cut-offs like tolerant graphics.