Improving the `bodyfile` format: a suggestion

Jan Starke <jan.starke@t-systems.com>

April 3, 2022

Abstract

In this paper we describe the famous `bodyfile` format and, especially, a lot of its shortcomings. But because this format is widely known and used, we discuss the benefits of `bodyfile` and take a look at the requirements that an improved fileformat should fulfill.

1 The `bodyfile` format

A lot of forensic tools collect data which are related with time information. We can see sich information as sets of events. Those data must be related to each other, which normally means that they are ordered chronologically, so that one can see which event occured after another. Unfortunately, a lot of events are extracted from many different sources, such as various filesystems, the Windows Registry, several log files and so on. Common tools which work with the `bodyfile` format include

- `fls`, which is part of The Sleuth Kit (Carrier (2003-2020))
- `evtx2bodyfile` can be used to convert Windows Event Log (`evtx`) files into `bodyfile`
- `mactime` is the standard to which is used to sort `bodyfile`s and generate human readable text- as well as csv-files
- `plaso`, which has its own SQLite based timeline format, is also able to consume `bodyfile`s

So, it seems that we have a common standard of how to describe events and their relation in time. It seems that we can use this standard to correlate a lot of different event sources to find interdependencies between events from different sources. It seems that because of the broad use, `bodyfile` is as base for sound forensic work. We will see that this is not the case.

1.1 Format of `bodyfile` files

The `bodyfile` format is text based, which means that nearly every commandline tool and text editor can work with it. A file in this format consists of a list von lines, where each line describes a single file which may be found on the filesystem under analysis, together with some metainformation about the file as well as the timestamps related to the file. Listing 1 show some lines of a bodyfile. Each line consists of a list of fields which are separated by the pipe character. The following fields are being used:

`MD5` MD5 hashsum of the file, or the number 0

`name` name of the file. Normally the fully qualified path is specified here

`inode` inode number or, on systems where no inodes are being used

`mode` describes what kind of a file this is and *who* is permitted to do *what* with this file. This field must be specified in the common unixoid syntax, e.g. `r/rwxr-xr-x` specifies this is a `r`egular file (`d` would denote a `d`irectory), that the owner of the file is allowed to read (`r`), write (`w`) and execute (`x`) the file, while members of the owning group as well as all others may read or execute the file, but are not allowed to write it. This field only makes sense when the filesystem has such a simple permission theme. As far as it comes to access control lists, this syntax is not sufficient anymore.

`UID` (numerical) identifier of the files owner, or 0 if this cannot be specified. For example, Windows dos not have numerical user identifiers, but uses Security Identifiers (SecurityIdentifiers) to identify users.

`GID` (numerical) identifier of the group which owns the file. the same restriction as with the *UID* applies here as well.

`size` size of the file in bytes

`atime` specifies at which time the file has lastly been *accessed*, as epoch time (number of seconds since January 1st, 1970). It is possible that this fields contains -1, which means that this value must be ignored. The same also applies to the other three timestamps.

`mtime` time of the last *modification* of the file, as epoch time

`ctime` time of the last *change* to the file's *meta-information*, as epoch time

crtime time of the files *creation* (birthtime), as epoch time

1.2 What's wrong with `bodyfile`?

A lot of people have found one or more simple problems which occured during their work with the `bodyfile` format. Fortunately, a lot of those problems are collected at the forensikswiki page (Metz (2021a)):

1.2.1 Missing timezone information

Despite a `bodyfile` entry has multiple timestamps, there is no information about from which timezone these timestamps has been collected. Also, there is no information if daylight savings should be applied. Most unixoid systems store their timestamps in Universal Time Coordinated (UTC), but it is – especially in forensic software – bad practice to silently assume such information. There are a lot of timestamp formats (e.g. Clyne and Newman (2002)) which also contain a timezone information and should better be used.

Why is this a problem? Imaging an analyst has two different event sources which must be correlated. Then a simple approach was to generate a `bodyfile` from each of those sources, join them together and sort them using `mactime`. This does only generate a useful result if both event sources use the same timezone or if the generator of the `bodyfile`s know how to normalize the timezones. Because the `bodyfile` does not contain timezone information, `mactime` will not be able to take this into account.

1.2.2 Unclear file encoding

When `bodyfile` was created, textfiles where basically ASCII encoded files. Also, filesystems used ASCII to encode filenames, so there was no need to support other encodings. But modern operating system support a lot of different encoding to be efficiently used by native speakers all over the world. So, a textfile now could be encoded in ASCII, UTF-8, UTF-16LE or in any other single- or multi-byte encoding. This makes it hard to correctly interpret filenames in the `bodyfile`.

1.2.3 Missing origin information

One could think that origin of the event information was clear: It was (mostly) the filesystem. But modern filesystems have multiple information about one single file, which might not identitical. For example, an NTFS filesystem could contain the following information about a file:

`$STANDARD_INFORMATION` is an attribute which contains the commonly used timestamps, which would be used in a `bodyfile`

`$FILE_NAME` attributes contain the name of the file, as well as an additional set of four timestamps. Furthermore, one single can have multiple `$FILE_NAME` attributes, e.g. to specify a short (8.3) name (`FILE_NAME_DOS`) in addition to its complete filename (`FILE_NAME_NTFS`). So, the Directory `Documents and Settings` contains an additional `$FILE_NAME` attribute with the name `Docume~1`. Each of those `$FILE_NAME` attribute has its own set of modified/accessed/changed/birth timestamps (Carrier, 2005, p.308-309).

`$UsnJrnl` (Update sequence number journal) is a special system file which tracks changes to file metadata. Every change is connected to a timestamp, so this file can also be the origin of filesystems events. Typically, events like movement or name changes are tracked here.

Historically, `fls` and also the `bodyfile` format have been dedicated to unixoid operating systems (and fileystems), and had sufficient functionality for this use case. To adapt to newer filesystems, `fls` has maxed out the capabilities of the `bodyfile` format. So, to enable handling of `$FILE_NAME` attributes, `fls` creates an additional entry for every file, and adds the string `($FILE_NAME)` as suffix to the `filename` field. We will see further situations where the atomicity of the `filename` field was broken, because `bodyfile` lacked the missing field.

1.2.4 Handling of links breaks atomicity

Most filesystems support one or more types of links. Unixoid filesystems have a notion of *hard link*, which is an additional inode that points to the same data, and *symbolic link*, which is an inode that points to another inode (roughly speaking). Technically, using hard links it's not possible to tell which of the links was the "original" file, so hard links are displayed as distinct files. When there is a symbolic link, say `file_b.txt` which points to `file_a.txt`, `fls` knows that this is a symbolic link, an appends the suffix `->` *`symbolic_link_target`* to the `filename` field. Thus, in our example, the `filename` field of `file_b.txt` would contain the value `file_b.txt -> file_a.txt`.

But there are more notions of links in other filesystems. For example, NTFS supports a third link type called *junction* (HardLinksAndJunctions). Neither NTFS symbolic links nor junctions are currently supported by `fls` (Metz, 2021b). However, there is also no support for such an information in the `bodyfile` format.

1.2.5 Unclear escaping policy

As already described, `bodyfile` uses the pipe character (`|`) as field separator. One might wondering which filesystems allowed the use of the pipe character in its

```
0|/home|524289|d/drwxr-xr-x|0|0|4096|1647870133|1646662575|1646662575|1646662311
0|/lost+found|11|d/drwx------|0|0|16384|1646662270|1646662270|1646662270|1646662270
0|/boot|19|d/drwxr-xr-x|0|0|4096|1647874025|1647874069|1647874069|1646662270
0|/boot/efi|20|d/drwxrwxr-x|0|0|4096|1648053561|1647873589|1647873589|1646662270
0|/boot/grub|21|d/drwxr-xr-x|0|0|4096|1646809126|1647873589|1647873589|1646662311
0|/boot/grub/gfxblacklist.txt|31|r/rrw-r--r--|0|0|712|1645606258|1645606258|1647873587|1646662312
```

Listing 1: Sample of a `bodyfile`

filenames. But keep in mind that there many different possible event sources. For example, one could use some log file which tracks command executions, as event source. As a command could contain a pipe character, the bodyfile would so, too. As this character could also appear in the contents of some field, this would make the parsing of `bodyfile` lines difficult. The current implementation of `mactime` is not able to handle `bodyfile` lines with pipe characters in one or more of its fields.

2 Alternatives to `bodyfile`

2.1 Timeline

2.2 Plaso & SQLite

2.3 elasticsearch

3 Desired properties of timeline formats

References

B. Carrier. The Sleuth Kit website. `https://www.sleuthkit.org/sleuthkit/`, 2003-2020. [Online; accessed 02-April-2022].

B. Carrier. *File System Forensic Analysis*. Addison-Wesley, 2005. ISBN 9780321268174.

G. Clyne and C. Newman. Date and Time on the Internet: Timestamps. RFC 3339, RFC Editor, July 2002. URL `https://www.rfc-editor.org/rfc/rfc3339.txt`.

HardLinksAndJunctions. Hard Links and Junctions. `https://docs.microsoft.com/en-us/windows/win32/fileio/hard-links-and-junctions`, 2021. [Online; accessed 03-April-2022].

J. Metz. Bodyfile. `https://forensicswiki.xyz/wiki/index.php?title=Bodyfile`, 2021a. [Online; accessed 02-April-2022].

J. Metz. fls: NTFS not showing symbolic link target and correct mode_as_string value. `https://github.com/sleuthkit/sleuthkit/issues/2645`, 2021b. [Online; accessed 03-April-2022].

SecurityIdentifiers. Security identifiers. `https://docs.microsoft.com/en-us/windows/security/identity-protection/access-control/security-identifiers`, 2021. [Online; accessed 02-April-2022].