Digital Investigation 16 (2016) 12–18

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

GVFS metadata: Shellbags for Linux

Christopher John Lees

Greater Manchester Police, DIU, Bradford Park, Bank Street Clayton, Manchester, Greater Manchester M320BL, United Kingdom

ARTICLE INFO

Article history:
Received 24 February 2015
Received in revised form 17 July 2015
Accepted 11 November 2015
Available online xxx

Keywords:
Linux
Forensic
GVFS
Encryption
Volume
Gnome
Computer
Investigation
Shellbag

ABSTRACT

There are a number of techniques that the perpetrator of an offence may use to hide data. These techniques include storing data on external devices or within encrypted containers. Although there are a number of recorded artefacts for the Windows operating system which may prove this, there is less information for artefacts for the Linux operating system. The Gnome Virtual File System produces files that relate to specific volumes and contain information about files stored within the volume, whether external device or encrypted volume. Examination of these files provides the potential to identify the names of files accessed, as well as the last accessed time of the files. This paper establishes some rules of when a filename is recorded in the metadata files and what data is recorded when the file is deleted, which can provide potentially useful information.

Introduction

Computer forensics has become an important part of investigations for a wide variety of crime. These range from the traditional crimes such as fraud through online websites and distributing Indecent Images of Children via the Internet, to newer offences such as hacking. Where the offence relates to the possession of certain material, such as Indecent Images of Children or terrorist documents, it is desirable for those who have such material to make it inaccessible to law enforcement agencies.

Encryption can be used to prevent access to files or hard disks by use of an encryption algorithm, making the data unreadable without access using a key and/or password. An encrypted volume, such as those created by the Truecrypt encryption software, can store multiple files within a single encrypted file. However, encryption can be easily detected due to the nature of the files (Garfinkel.S, 2007).

As such, storing encryption on external media such as a pen drive or external hard disk can allow the perpetrator of such offences to hide the device with the encrypted files(s) whilst leaving their computer in plain sight.

Under some versions of the Windows operating system (Windows Vista and later) there are forensic artefacts left behind by opening encrypted volumes, including the “Shell bag” entries and the Windows Search Index (in the Windows.edb) file. In some circumstances, these artefacts document the files and folders contained on external media devices and even in encrypted volumes.

However, under the Linux operating system these artefacts are not present and another way of obtaining information about mounted devices (including encrypted volumes) may be possible through the Gnome Virtual Filesystem (GVFS) metadata.

Introduction to the GVFS

The GVFS was introduced in Gnome desktop version 2.22 and was a replacement for the GNOME-VFS. The GVFS

E-mail address: Christopher.Lees@GMP.police.uk.

http://dx.doi.org/10.1016/j.diin.2015.11.001
1742-2876/

runs a process (daemon) in the background which keeps a track of file systems mounted using the GVFS (Gnome Help, 2008). The GIO API allows developers to access data stored by the GVFS process.

Related studies

Gnome desktop system

GNOME is a graphical user interface which includes a desktop environment which works with many Unix type operating systems, including GNU/Linux.

By default, Ubuntu 14.04 uses the Unity desktop system, which is a shell interface for the GNOME desktop environment. Unixmen.com (2013) states that Ubuntu is the most popular distribution of Linux and although this source states that the actual number of users cannot be determined for definite, it gives a figure of 20 million users of the OS.

According to Gnome Help (2008), version 2.22 of the Gnome introduced GVFS as a replacement for the Gnome-VFS system. GVFS is described as a single master daemon (named gvfsd) and separate daemons for each GVFS mount and the master daemon keeps track of the different mounts.

GVFS metadata files

Stackoverflow (2012) states that the metadata stored by the GVFS is located in the ~/.local/share/gvfs-metadata folder, and that each partition will have its own name which relates to the partition. According to AskUbuntu (2012) the Universally Unique Identifier (UUID, see Section 2.3) of the partition is used to name the file. However, no details are given for partitions which are not EXT based, such as FAT or NTFS.

It was not possible to find sufficient information relating to the GVFS metadata from sources of information such as academic papers or documentation. However, the source code for the Gnome desktop manager is available as it is an open source project.

The explanatory notes provided within the GVFS source repository, shown in GitHub (2009), gives detail about the structure of the metatree when stored in a file and shows the following structure for the MetaFileHeader under the heading “generic”:

Breath-first stored, first tree, then data
offsets and sizes are uint32
time_t are uint32 with base stored in header
data stored in big endian
non-string blocks padded to 32bit
all key names and values are utf8, without zeros
filenames are byte strings

It then proceeds to give a more detailed description of the file format, which has been summarised below:

The file starts with a header section which includes a magic number, an offset to the root node and keyword entries and a base timestamp. The magic number is defined with the metatree.c source code file as “\xda\x1ameta”. The definitions seem to relate to the “magic” field described in the text file. As such, the file signature identifier for the file would be 6 bytes long and consists of the hex DA 1A followed by the text “meta”.

Following this, the major and minor versions are a single character each, which would therefore take the next 2 bytes. Then there are 2 × 32 bit (8 bytes total) integers prior to the offset to the root entry. The offset to the root entry would therefore be at offset 16 of the file and 4 bytes long. Following this is a 4 byte offset to the keyword entries, followed by an 8 bytes timestamp.

The line “*time_t are uint32 with base stored in header*” from the generic description of the header appears to show that the time stored here is the base time, and any times stored in entries is relative to this value.

The structure described above matches a structure named MataFileHeader which is defined within the code and shown below:

guchar magic[6];
Guchar major;
Guchar minor;
Guint32 rotated;
Guint32 random_tag;
Guint32 root;
Guint32 attributes
Guint64 time_t_base;

The root directory entry is described as 3 offsets and a timestamp. The first offset is to the directory name, which for the root of a directory would always be “/”, the second is the offset to the child entries and the third is the offset to the metadata entry.

The child entry is shown as starting with an integer for the number of child entries contained followed by an array for each of the child objects. This array contains the offset to the filename of the entry, the offset to the items child object(s) and the offset to the metatdata for this entry. After the array is an array of null terminated strings which contains the filenames of the child entries.

The metadata entry is shown to start with an integer containing the number of keys, followed by an array of keys. The array consists of a integer named “keyword”, with a note that if high bit is set it is a list, and an offset value to the string or array of strings. This array is followed by a block of string arrays which are referred to by offsets in the array.

The keyword entry appears to start with an integer showing the number of keywords, followed by an array of offsets and a string block which contains the offsets stored in the array.

Although the details of the stored data are shown, there is no information relating to under what circumstances file names and metadata are included in the meta file.

Using the information provided about the structure of the GVFS metadata file, a program was developed to interpret the data and the source code for this can be seen in Appendix A.

As well as the main metadata file, a second file is also described which is described as a journal file for the met-

adata file. The purpose of this file appears to be to store updates to the metadata information temporarily. Once the updates have been applied to the metadata file, the journal is emptied.

The structure of this file is also described within the file-format.txt file. The file starts with a journal header comprising a 6 byte magic header, followed by a 2 byte character version. After this there are three 32 bit integers, the first is named "random_tag", the second is the file size and the third is the number of entries.

After the header there is a structure described for the journal entry. This starts with a integer containing the entry size and followed by a crc32 checksum of the data in the entry. Following this is a 64 bit timestamp and a single byte denoting the type of operation, the type of allowed data is shown as "set: 0, set_list: 1: unset: 2, move: 3, copy: 4". If the operation is a copy or move, then the target is stored as a string after the operation byte.

The data included after this varies depending on the operation type. The set operation stores 2 strings, a key and a value. The set list operation stores a key as a string, a 32 bit integer for the number of values and a further string of values. The unset operation stores the key string only and the copy contains the source path as a string.

Regardless of the operation the entry size is recorded at the end of the record. The notes for this state it must be the same as the size at the beginning of the entry.

EXT4 file system

EXT4 is the fourth extended file system, superseding EXT3, and is a journaling file system used by the Linux operating systems. The file system was released into the Linux kernel on the 11th of October, 2008 and provided a number of improvements to the EXT3 file system, including support for volumes up to 16 tebibytes in size, and journal checksumming to increase journal reliability. It is also backwards compatible with the EXT3 and EXT2 file systems.

An EXT4 file system has a superblock which is located at offset 1024 of the partition. This superblock contains information about the partition, including size and location of journal, etc. It also contains a Universally Unique Identifier (UUID), which can be used to identify the partition. The UUID is a 128 bit number, which is represented as 32 hexadecimal digits and is stored at offset 0x68 of the EXT4 superblock (EXT4 Wiki, 2015). Fig. 1 shows a superblock with the UUID section highlighted which is "4adb2e3c6c7344e9a4c85781b408ff91".

Truecrypt

The Truecrypt software is an open source implementation which allows a user to create on-the-fly encrypted volumes. The latest version of the Truecrypt software at the time of writing was 7.2, however version 7.1a was the last version that allowed creation of encrypted volumes. On-the-fly means that the data is decrypted when it is accessed, and encrypted when it is saved without a user first having to manually decrypt it. That is to say, that once an encrypted volume is opened (mounted) on a system, data can be read and written to the volume as if it were a standard drive on the computer. Once the volume is closed (dismounted) all the data within the volume is encrypted and inaccessible without the password and/or keyfile(s). When an encrypted volume is mounted, it is allocated a drive letter (under the windows operating system) or under the Linux operating system, a folder is created in the "/media" folder with the name "truecrypt*x*" where the "*x*" is a number which corresponds to the slot in which the volume can be mounted.

Methodology

Objective

The objective of the subsequent experiments is to determine what information is held within the GVFS-metadata files, and under what circumstances it is created. It will also examine the file naming convention for files relating to different file system types, including NTFS, FAT and EXT based file systems.

Research methodology

Taking into account the above objective, examination of the literature review provided little information on when the GVFS metadata is created and what information is contained. As such, the research into this area was approached through experimentation. It is the author's view that this was the best method to identify the changes made to the GVFS metadata when actions are applied to the mounted volume, whether encrypted or not.

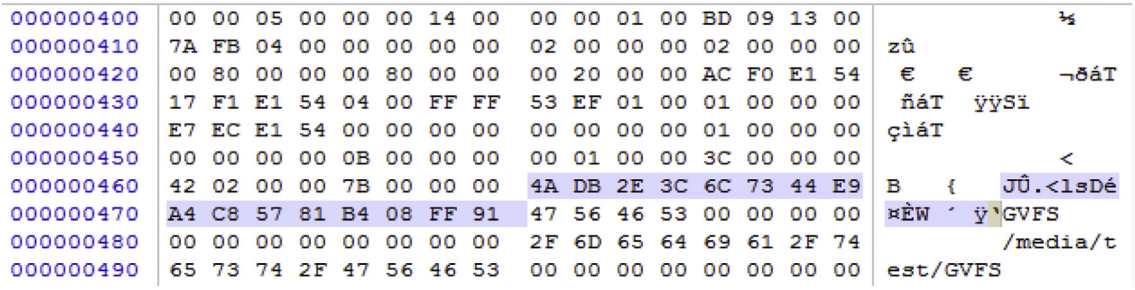

Fig. 1. UUID within superblock.

Experimental setup

In order to carry out the experiments in line with the objective, a Linux system was required. Given that the literature review suggested that the Ubuntu distribution is widely used, this was installed on the system. At the time of writing, 14.04 was the latest version of this distribution and is available in 32 and 64 bit versions, and the 64 bit version was used throughout these experiments. The system was a virtual machine, which will be created using VMware Workstation 10.

An external drive and an encrypted container, including test files and folders, was required for the experiments. For the external device, a second virtual hard disk was created in VMware Workstation and this was attached to the Linux system. Prior to this, the test data was loaded onto the virtual disk using another Linux machine. This was to prevent creation of any data on the test system prior to the experiments. Both of these storage areas was formatted using the EXT4 file system.

Both the Truecrypt volume and the virtual hard disk contained the same files and folder structure and underwent the same actions in order to ascertain any differences. The test data consisted of the source code from the Libewf website, specifically the download "Libewf-20140608", the source code for Foremost (version 1.5.7), and the "8-jpeg-search" test images from the Digital Forensics Tool Testing website.

This data provided a variety of file types which included files which are easily viewable such as, text and html files, and others which were not, such as raw image files. It also provided a folder structure which had multiple layers and a number of files in each folder.

Further virtual disks were created which were formatted as NTFS and FAT32. These contained the same data as the earlier storage areas, but were only used to determine the file naming convention for non-EXT based file systems.

Experiments

In order to determine the information required for the objective, a number of experiments were required. There were two distinct strands of experiments, those involving a physical mounted device (a virtual disk), and those involving a mounted Truecrypt volume.

For each of these, the following points were established:

- What information is stored in the meta data file when a volume is mounted?
- What information is stored when a file is accessed?
- What information is stored about other files in a folder where a file is accessed?
- Is the time information updated on subsequent file accesses?
- What information is stored when a file is deleted?

In addition to the above points, the naming convention of the files needed to be determined to see if the names can be linked back to the volume they originated from. This included testing file systems which did not contain a UUID such as, FAT32 and NTFS.

In order to address these points a number of experiments were devised. Throughout these experiments, it needed to be confirmed how the time is stored within the file. The related studies section suggested that all times in the file were stored as offsets to the time in the header. This was confirmed by noting the times and dates of actions and comparing these to the offset and timestamp located in the metadata files.

Experiment 1 – metadata file naming convention

This experiment was designed to determine the link between the filename of the metadata file and the file system. As mentioned in the literature review, it has been suggested that the metadata files use the UUID of the EXT volume for the file name, however the FAT32 and NTFS file systems do not use a UUID.

In this experiment, 3 volumes of different file types were created; EXT4, FAT32 and NTFS, and attached to the virtual machine. Each of these had files on which were unique to their volume so that the metadata files which were created could be linked back to the volume. Once the metadata files were linked to their correct volume, the filenames were examined to determine if there was a link between the volume and the filename.

After the completion of the experiment, the virtual machine was reset to its original state as it was prior to this experiment.

Experiment 2 – mounting device

This experiment was designed to determine the effect on the metafile(s) when a device is attached to the system, but not accessed, i.e. no files were opened. In order to test this, both the virtual disk and the encrypted volume were attached to the virtual machine, and were mounted by the Ubuntu operating system. Once this has been done, the virtual machine was left to allow any data to write for a minimum of 10 minutes and following this, the system was shut down. Once the system was shut down, the metafile(s) were extracted from the virtual hard disk using FTK Imager.

This file was then decoded using the information shown Appendix A of this document.

Experiment 3 – file access

This experiment was designed to determine the effects on the metafile(s) when a device was attached to the system and various file accesses took place. In this experiment, the system was started and the virtual disk and encrypted volumes were connected. Once the file systems were mounted, a variety of file accesses was made. The times of these accesses were recorded for comparison to any dates and times located in the metadata files.

The types of file access included:

- Accessing a file which is contained within a folder which includes other files.
- Accessing a file contained within a subfolder of another folder. The subfolder will contain additional files to those file(s) accessed.

- Accessing a file using a terminal as opposed to the Nautilus file manager.
- Accessing a file which has no default editor defined (such as a raw image file).
- Using automated file access such as a virus scanner. Clamav was used for this.

Once the relevant activity was completed, and enough time had been allowed for the data to be written, the virtual machine was shut down and the metadata file(s) was extracted from the virtual hard disk using the FTK Imager software.

Experiment 4 – file deletion

This experiment was designed to determine the effect on the meta file when a file which had been previously included in the metafile was deleted.

In this experiment, a file which had been previously accessed in Experiment 2 was chosen for deletion. The file was deleted using the Nautilus file manager by pressing the delete key which should have moved the file to the "Trash" folder. The system was then left for 10 minutes to allow any data to write and the machine was then shut down. The metadata file(s) was then extracted from the virtual disk using FTK Imager. Following, the system was restarted and the file which was now located in the "Trash" folder was deleted by emptying the "Trash" folder. The system was left for 10 minutes to allow any data to write and the machine was be shut down. The metadata file(s) was then extracted from the virtual disk using FTK Imager.

This should have been able to show the differences between files which were moved to the "Trash" folder as opposed to those which were deleted.

Results

The objective of the experiments was to determine what information was held within the GVFS-metadata files, and under what circumstances it was created, as well as determining if the metadata file name could be used to link the file to a specific volume. The results from the 4 tests are shown below.

Experiment 1 – metadata file naming convention

In Experiment 1, different type of file systems were connected to the virtual machine and files were accessed in order to create metadata files for the volumes. Once the metadata files were created, the filenames were examined to determine if they could be linked to the volume.

The metadata file created for the EXT4 volume was named "uuid-4adb2e3c-6c73-44e9-a4c8-5781b408ff91". Examination of the EXT4 superblock for this volume showed that the UUID of the volume was "4adb2e3c6-c7344e9a4c85781b408ff91". Therefore, to link a EXT4 partition to its metadata file, a file name containing the UUID of the volume was be located.

The metadata file created for the FAT32 volume was named "uuid-F8C2-BE3C". Examination of the FAT32 boot sector, using the structure show by Carrier (2005), showed that the volume serial (in hex) was 3CBEC2F8. This can be found at offset 67 of the FAT32 boot sector, just before the Volume label field. Therefore, in order to link a FAT32 file system to its metadata file, a filename with the reversed volume serial should be located.

The metadata file created for the NTFS volume was named "uuid-1206B9EF06B9D3C7". Examination of the NTFS boot sector showed that the volume serial (64bit hex) was C7D3B906EFB90612. This can be found at offset 72 of the NTFS boot sector. Therefore, in order to link a NTFS file system to its metadata file, a filename with the reversed 64bit volume serial should be located.

Experiment 2 – mounting device

In Experiment 2, the file systems were mounted to see what metadata files were created by mounting the volume.

In the case of the external mounted device, no metadata files were created from the file system being mounted. Even when the volume was accessed, without any file access, no metadata files were created. This was the same result for both the virtual disk and the Truecrypt volume.

Experiment 3 – file access

In Experiment 3, various files were accessed in the volumes to determine what information was stored in the metadata files and under what circumstances.

The first area of examination was accessing files within folders, sub-folders and folders with multiple files in to determine under what circumstances file names were stored in the metadata. The results showed that file names were not stored in the metadata until the files had been accessed, and folder names only appeared when a file within a folder or sub-folder had been accessed. Only filenames that were accessed were stored in the metadata; none of the other files located in the same folder as the accessed file were recorded. Therefore, the name of the file existing in the metadata would be consistent with a user accessing that file.

When accessing files using the terminal as opposed to the file manager, the results were the same; the filename was still recorded in the metadata. However, when the clamav antivirus software was used to scan all of the files in the volume; no further entries were recorded. This appeared to show that only user activity caused the entry to be added to the metadata, as opposed to automatically accessed.

When a file was accessed which could not be opened natively, in this case a raw DD image, the filename was not added to the metadata. This was true even if a program was selected to open the file with, Ghex in this instance, and also held if that software was set to automatically open files of that type.

It was noted that throughout the experiment, the times in the metadata were stored in UTC, regardless of the time zone the OS was set to, and the time was stored as an offset from the timestamp located in the header of the metadata file. This timestamp was created at the time of the first file access to the volume.

The results were identical for the virtual disk and the Truecrypt volume.

Experiment 4 – file deletion

In Experiment 4, the files were deleted in order to ascertain what changes this would make to the metadata file.

When a file was deleted, or rather moved to the "Trash" folder, the metadata updated to reflect that the file had moved. When the Trash folder was emptied the metadata did not change. This suggested that the names of files which were deleted from a volume would be visible in the metadata, specifically the "Trash" folder.

The results were identical for the virtual disk and the Truecrypt volume.

Conclusions

After carrying out the experiments, it can be seen that the GVFS metadata files can contain information about files which are located on an external device and/or encrypted containers. However, merely mounting the device did not create a metadata file, and files had to be accessed from the volume for this to happen.

It also only appeared to show the file names for the file which the user has accessed as opposed to the entire file listing, and also deleted files are recorded as being in the "Trash" folder. Files accessed through automated access, by the anti-virus scanner, did not get included in the metadata files. Files which were accessed outside of the standard file manager, in this case by terminal, were still recorded in the metadata file.

Where a file was not viewable using the built-in software, the filename did not appear in the metadata. This held even if a viewer program was selected and/or a default program was set for that file type.

The experiments confirmed the theory from the literature review about how the time details were stored within the metadata files.

Conclusion

This section contains a summary of the conclusions found during the tests, as well as an evaluation of the methodology and tools used.

Conclusions from tests

From the results, it is clear that the GVFS metadata files contained information that may be significant to forensic computer investigations.

The metadata files have the potential to store information about files which have been accessed on external devices and encrypted containers and also provides a way of linking the metadata file to a specific volume. The naming of the file provides information about the file system it relates to as NTFS, EXT4 and FAT32 file systems all have a different naming convention.

These files also have the potential to show the names of files which have been deleted from the system and show a time the file was last accessed. The tests also show that scanning the files using the "Clamav" anti-virus scanner did not put information about the files scanned into the metadata files.

Evaluation of methodology and tools

The overall methodology used for these experiments worked well. The virtual machines provided an excellent way to create the data and allowed the system to be "rolled back" to an earlier point in time if any mistakes were made. It was noted that in many of the experiments, turning the virtual machine off to extract the data was an unnecessary step; the data could be extracted from the folder on the virtual machine by 'dragging' it out to the host machine. In this case, it was important to allow enough time for the metadata files to be updated, in these experiments 5 minutes was allowed before attempting to access the metadata file.

The program shown in Appendix A provided an easy way of interpreting the metadata files and the results of the program were consistent with the activity that had taken place. Although the program extracted the metadata information, it provided no interpretation of this data which could be an area of further work.

Further work

Although this work has provided a method of identifying under what circumstances data was put into the metadata files, there are a number of areas of further work. These include:

- Examining any differences in behaviour between file systems (NTFS, FAT32) and the data stored in the GVFS metadata files.
- Examine the naming and other properties of metadata files for other file systems, including XFS and BTRFS and also for pluggable file systems such as SFTP, SMB and MPT.
- Throughout the experiments, the file name was the only data interpreted from the metadata files. Interpreting the metadata information recorded in the file, rather than extracting the filename and folder structure could provide further useful information.
- Examining any time constraints on how long the metadata is stored for.
- Examining the metadata log files to interpret any data stored in these files.

Appendix A. Code for interpretation of GVFS metadata

Source code can be found at: https://github.com/minime2k10/GVFS-Metadata.

References

AskUbuntu, How. 'How does ubuntu know that i have a folder icon customized?'. Askubuntu. N.p., 2012. Web. 17 Feb. 2015. Available

from: http://askubuntu.com/questions/171077/how-does-ubuntu-know-that-i-have-a-folder-icon-customized.

Carrier B. File system forensic analysis. Boston, Mass: Addison-Wesley; 2005.

EXT4 Wiki., 'Ext4 Disk Layout - Ext4'. N.p., 2015. Web. 16 Feb. 2015.Available from: https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#The_Super_Block.

Simson Garfinkel, "*Anti-Forensics: Techniques, Detection and Countermeasures*", 2nd International Conference in i-Warefare and Security, pp 79, 2007.

GitHub., 'GNOME/Gvfs'. N.p., 2009. Web. 17 Feb. 2015. Available from: https://github.com/GNOME/gvfs/blob/master/metadata/file-format.txt.

Gnome Help., 'GNOME 2.22 Release Notes'. N.p., 2008. Web. 16 Feb. 2015. Available from: https://help.gnome.org/misc/release-notes/2.22/#sect:gvfs-gio.

Stackoverflow, GNOME: 'GNOME: Where Does Nautilus Store Emblem Data And How?'. Stackoverflow. N.p., 2012. Web. 17 Feb. 2015. Available from: http://stackoverflow.com/questions/10874702/gnome-where-does-nautilus-store-emblem-data-and-how.