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 10 December 2015

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-