

TFS4-Hidden Protected Area

Application Note

October-2007, Version 1.0





Copyright Notice

Copyright © 2007, Flash Software Group, Samsung Electronics Co., Ltd

All rights reserved.

Trademarks

TFS4 is a trademark of Memory Division, Samsung Electronics Co., Ltd in Korea and other countries.

Restrictions on Use and Transfer

No part of this guide may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Samsung Electronics Co., Ltd. as governed by Republic of Korea and international copyright laws.

Document confidentiality status: Restricted Distribution

Readers of this document are not granted the right to distribute this document to any third party without prior written agreement from Samsung Electronics Co., Ltd.

Samsung Electronics Co., Ltd may make changes to the contents of this document any time without notice.

THIS DOCUMENT IS INTENDED ONLY TO ASSIST THE READER IN THE USE OF THE PRODUCT. SAMSUNG ELECTRONICS Co., Ltd. SHALL NOT BE LIABLE FOR ANY LOSS OR DAMAGE ARISING FROM THE USE OF ANY INFORMATION IN THIS DOCUMENT. SAMSUNG ELECTRONICS Co., Ltd. ASSUMES NO RESPONSIBILITY FOR ANY ERROR OR OMISSION IN INFORMATION CONTAINED IN THIS DOCUMENT, OR ANY INCORRECT USE OF THE PRODUCT.

Contact Information

Flash Software Group
Memory Division
Samsung Electronics Co., Ltd

Address: San #16 Banwol-Dong, Taeon-Eup
Hwasung-City, Gyeonggi-Do, Korea, 445-701



Summary

This paper focuses on hidden areas of flash drive, specifically Hidden Protected Area. Hidden area is a directory that is not shown to the other FAT compatible filesystem. The Hidden Protected Area (HPA) as defined is a reserved area on the data region of a flash Drive. It was designed to store information in such a way that it cannot be easily modified, changed, or accessed by the user. This paper presents an overview and an in-depth description of the new hidden protected area (HPA) based on internal NAND solution. The present invention employs the idea of Bad Block concept to prevent reading or writing of data in to the HPA by the end user. The Samsung's NAND solution will come with the HPA-based solution. The hidden protected area, enables Samsung to provide a recovery solution that provides greater flexibility and that enhances the security for recovery data.

Description

TFS4 FAT file system is composed of four different sections.

The Reserved sectors located at the very beginning. The first reserved sector is the Boot Sector.

The FAT Region. While data is stored in 512-byte sectors on the flash drive, for performance reasons individual sectors are not normally allocated to files. The reason is that it would take a lot of overhead (time and space) to keep track of pieces of files that were this small. The flash drive is instead broken into larger pieces called clusters, or alternatively, allocation units. Each cluster contains a number of sectors. Typically, clusters range in size from 2,048 bytes to 32,768 bytes, which corresponds to 4 to 64 sectors each.

The file allocation table is where information about clusters is stored. Each cluster has an entry in the FAT that describes how it used. This is what tells the operating system which parts of the flash drive are currently used by files, and which are free for use. The FAT entries are used by the operating system to chain together clusters to form files.

Each entry records one of five things:

- the cluster number of the next cluster in a chain

- a special end of clusterchain (EOC) entry that indicates the end of a chain
- a special entry to mark a bad cluster
- a special entry to mark HPA
- a zero to note that the cluster is unused

HPA entry is created at the last sector of the FAT area. If it's not available then HPA cannot be created.

The FAT region contains two copies of the File Allocation Table. The file allocation tables are stored in the area of the drive immediately following the volume boot sector. Each volume actually contains two identical copies of the FAT; ostensibly, the second one is meant to be a backup of sorts in case of any damage to the first copy. Damage to the FAT can of course result in data loss since this is where the record is kept of which parts of the disk contain which files. The problem with this built-in backup is that the two copies are kept right next to each other on the disk, so that in the event that for example, bad sectors develop on the disk where the first copy of the FAT is stored, chances are pretty good that the second copy will be affected as well.

Cluster sizing (and hence partition or volume size, since they are directly related) has an important impact on performance and disk utilization. The cluster size is determined when the disk volume is partitioned. Certain utilities can alter the cluster size of an existing partition (within limits) but for the most part, once the partition size is selected it is fixed.

The Root Directory Region. This is a Directory Table that stores information about the files and directories located in the root directory. It is only used with FAT12 and FAT16 and means that the root directory has a fixed maximum size which is pre-allocated at creation of this volume. FAT32 stores the root directory in the Data Region along with files and other directories instead, allowing it to grow without such a restraint.

The Data Region. This is where the actual file and directory data is stored and takes up most of the partition. The size of files and subdirectories can be increased arbitrarily (as long as there are free clusters) by simply adding more links to the file's chain in the FAT. The HPA area is allocated in this region. If the normal area is X%, the HPA area will be (100-X)%.



By separating data in the hidden protected area, this solution provides greater protection from the data loss and unauthorized access.

HPA User Configuration Setting

For Configuration of the HPA root directory, use the following command:

```
#define TFS4_HPA_ROOT_NAME
“__hidden__”
```

You can define the maximum volume size of HPA by the following macro:

```
#define TFS4_HPA_MAX_VOLUME_SIZE
1024 MB
```

You can define the maximum volume count of HPA by the following macro:

```
#define TFS4_HPA_MAX_VOLUME_COUNT
1
```

The files under the HPA root directory are hidden from the user. You can put the files in HPA by the following command:

```
TFS4_MOUNT_HPA
```

HPA is created when the mount command is given: `tfs4_mount()` with `TFS4_MOUNT_HPA` flag and there is no HPA on the volume.

```
TFS4_MOUNT_HPA_CREATE
```

It creates HPA structure on the volume when there is no hidden area. If user tries to access the files from the hidden directory, TFS4 returns error. To remove HPA area, formatting the volume is recommended.

Note: Hidden area root directory can be neither renamed nor removed from HPA to Normal area.

Using the File System

We take a look at a sample program here. The first section of the code performs mount, create, write & read operation of the file in the HPA area. The second half of the code performs the write operation in normal area. Understanding this sample will make writing your own FAT application easier.

```
t_int32 r, errno;
t_uint8 pBuffer[64*1024];

/* mount 1st partition of /dev/nf, attach to /a/, with
HPA */
r = tfs4_mount("/dev/nf0", "/a/", "KFATFS",
TFS4_MOUNT_HPA, 0);
if (r >= 0)
{
/* mount operation was success */

// this operation writes data in HPA.

szPath = "/a/__hidden__/HPA.txt";
dwFD = g_stTFS4api.tfs4_creat(szPath,
FILE_ARCHI_ATTR);
if(dwFD < 0)
goto error;

r = g_stTFS4api.tfs4_write(dwFD, pBuffer,
sizeof(pBuffer));
if(r < 0)
goto error;

/* read operation */

r = g_stTFS4api.tfs4_read(dwFD, pBuffer,
sizeof(pBuffer));
if(r < 0)
goto error;

r = g_stTFS4api.tfs4_close(dwFD);
if(r < 0)
goto error;

// this operation write some data in normal area.
szPath = "/a/normafile.txt";
dwFD = g_stTFS4api.tfs4_creat(szPath,
FILE_ARCHI_ATTR);
if(dwFD < 0)
goto error;
```



```

r = g_stTFS4api.tfs4_write(dwFD, pBuffer,
sizeof(pBuffer));
if(r < 0) goto error;

r = g_stTFS4api.tfs4_close(dwFD);
if(r < 0)
goto error;
...
}
else
{
// mount fail !!!
errno = tfs4_get_errno();
...
}

```

First, there are some variables declaration: a file structure to be used as a handle to the file we will be creating- dwFD and a buffer- pBuffer that will be used for reading and writing our file. The integer r is for the code returned by the TFS4 API functions. This code should always be checked to make sure whether the operation is success or not.

```

r = tfs4_mount("/dev/nf0", "/a", "KFATFS",
TFS4_MOUNT_HPA, 0);

```

This function mounts a partition onto a file system volume. tfs4_mount() attaches the filesystem specified by device to the volume specified by target.

The parameters are as follows:

- “/dev/nf0” points to the logical device name. Logical device name consists of physical device and partition number. “/dev/nf0” means first partition of physical device registered as “/dev/nf”.
- “/a/” refers to the target volume name
- KFATFS refers to the filesystem name
- TFS4_Mount_HPA It mounts a volume with hidden area. tfs4_mount() will fail with this flag when there is no hidden area on the volume.
- Null value

Next the sample program creates and opens a file. It returns a file descriptor

```

dwFD = tfs4_creat(szPath,
FILE_ARCHI_ATTR);

```

The parameters are as follows:

szPath path for the file to be created in HPA area

```

szPath = "/a/ __hidden__ / HPA.txt"

```

FILE_ARCHI_ATTR describes the file attribute.

Next the sample program writes the data stored in pBuffer into the file

```

tfs4_write(dwFD, pBuffer, sizeof(pBuffer));

```

The parameters are as follows:

- dwFD is file descriptor
- pBuffer A buffer that the written data is stored
- sizeof(pBuffer) number of characters to write

```

tfs4_read(dwFD, pBuffer, sizeof(pBuffer));

```

The function tfs4_read reads data from a dwFD file stream up to sizeof (pBuffer) bytes and stores it into a memory area that pBuffer points to. The length of the read data is converted into a byte unit. On success the number of the bytes read is returned and the file position is advanced by this number.

The file can now be closed by **tfs4_close(dwFD)**.

In order to write the data in normal area, similar operation is performed, but this time the file path is changed to the normal area.

Conclusion

This paper describes about the HPA in the Samsung’s TFS4 FAT based file system where the information is hidden to the end user. Only the vendor can access the region and decide the contents to be hidden to the end user.