Chapter 6

# ANTI-FORENSIC CAPACITY AND DETECTION RATING OF HIDDEN DATA IN THE Ext4 FILESYSTEM

Thomas Göbel and Harald Baier

**Abstract** The rise of cyber crime and the growing number of anti-forensic tools demand more research on combating anti-forensics. A prominent anti-forensic paradigm is the hiding of data at different abstraction layers, including the filesystem layer. This chapter evaluates various techniques for hiding data in the ext4 filesystem, which is commonly used by Android devices. The evaluation uses the capacity and detection rating metrics. Capacity reflects the quantity of data that can be concealed using a hiding technique. Detection rating is the difficulty of finding the concealed artifacts; specifically, the amount of effort required to discover the artifacts. Well-known data hiding techniques as well as new techniques proposed in this chapter are evaluated.

**Keywords:** Anti-forensics, data hiding, ext4, filesystem forensics

## 1. Introduction

The rise of cyber crime and the proliferation of anti-forensic software and tools that interfere with forensic investigations demand more research in the area of anti-forensics [15]. In 2007, Garfinkel [7] cautioned about the number of tools that were available to frustrate forensic investigations. More recently, Conlan et al. [4] have released a huge data set that includes 308 anti-forensic tools. While investigators generally take strong measures to keep digital evidence intact, malicious entities attempt to hide, remove, destroy or alter evidence, rendering forensic investigations difficult, time-consuming and expensive. Anti-forensics also deals with data hiding techniques [13]; a variety of artifact wiping tools and trail obfuscation methods are used to deliberately disorient forensic investigations.

© IFIP International Federation for Information Processing 2018
Published by Springer Nature Switzerland AG 2018. All Rights Reserved
G. Peterson and S. Shenoi (Eds.): Advances in Digital Forensics XIV, IFIP AICT 532, pp. 87–110, 2018.
https://doi.org/10.1007/978-3-319-99277-8_6

The popular ext4 filesystem is used as the default by Android and many Linux operating system distributions since kernel 2.6.28 [18]. It was created as the successor to the ext3 filesystem to keep up with increasing disk capacities and advanced features [10]. In the context of forensic investigations, it is important to know about the potential hiding places in ext4 volumes, especially due to the extensive use of ext4 in Android smartphones since version 2.3 Gingerbread [16].

Anderson et al. [1] conducted a study of data hiding in filesystem metadata and developed a steganographic filesystem. This resulted in the creation of StegFS [11], a steganographic filesystem based on ext2 that secured hidden data.

Various data hiding techniques for the ext2 and ext3 filesystems, as well as suitable countermeasures, are discussed in [3, 5, 12]. A security analyst named The Grugq [14] has developed several anti-forensic tools that hide data within ext2, but these tools have not been updated for newer versions of the ext filesystem. The most recent contribution is a low-level study and comprehensive forensic analysis of the most important ext4 filesystem data structures by Fairbanks [6]. This research also identified potential hiding places in ext4, such as HTree nodes, group descriptor growth blocks (GDGBs) and data structures in uninitialized block groups, but it did not study these hiding places any further.

Mathur et al. [10] have published extensive research related to the new ext4 filesystem. Wong [19] has created the *Ext4 Wiki*, an important reference for filesystem analysis. Both these references have been used in this research to locate possible hiding places in the ext4 filesystem. Essential information about the ext4 filesystem layout can be found in the source code of the Linux kernel (see, e.g., [17]).

Some academic research has focused on data hiding in previous versions of the ext filesystem. However, public research related to ext4 anti-forensics is practically non-existent.

This research focuses on the key question – How can data be hidden in an ext4 volume? In particular, the research differentiates between new methods and known techniques discussed in research related to previous ext filesystem versions, and studies their functionality in the context of ext4. New features and data structures of ext4 are analyzed and their efficacy in hiding data is evaluated; this helps identify specific filesystem structures that can be used to hide data. The data hiding methods are evaluated using two metrics: (i) capacity; and (ii) detection rating. Capacity expresses the amount of data that can be hidden whereas the detection rating expresses the difficulty of finding concealed artifacts (e.g., effort required on the part of forensic investigators). For this reason, ext4 volumes are examined using common, open-source forensic

*Table 1.* Overview of ext4 hiding techniques.

| Data Hiding Techniques | Filesystems |
|---|---|
| **Previous Techniques** | |
| File and Directory Slack Space [3, 5] | ext2/ext3 |
| Null Directory Entries [14] | ext2 |
| Partition Boot Sector [12] | ext2/ext3 |
| Superblock: Slack Space [3] | ext2/ext3 |
| Superblock: Reserved Space [14] | ext2 |
| Superblock: Backup Copies [12] | ext2/ext3 |
| Group Descriptor Table: Slack Space [3] | ext2/ext3 |
| Block Group Descriptor: Reserved Space [14] | ext2 |
| Inode Table: Reserved Inodes [5, 12] | ext2/ext3 |
| Inode: Reserved Space [14] | ext2 |
| **New Techniques** | |
| Block Bitmap: Slack Space [*] | ext4 |
| Inode Bitmap: Slack Space [*] | ext4 |
| Inode: Slack Space/Extended Attributes [*] | ext4 |
| Inode: Nanosecond Timestamps [*] | ext4 |
| Group Descriptor Table: Backup Copies [*] | ext4 |
| Group Descriptor Table: Growth Blocks [6] | ext4 |
| Extent: Persistent Preallocation [*] | ext4 |
| Data Structures in Uninitialized Block Groups [6] | ext4 |

tools such as FTK Imager, Autopsy and Sleuthkit. Finally, the research evaluates the forensic implications of the various data hiding methods.

Table 1 lists all the ext4 data hiding techniques evaluated in this research. The table is organized into two sections, one containing techniques applied to earlier versions of the ext filesystem and the other containing new or untested techniques for the ext4 filesystem. The new techniques, which are marked as [*], are the primary contributions of this research.

## 2. Background

This section discusses anti-forensics and the technical aspects of the ext4 filesystem.

### 2.1 Anti-Forensics

Baggili et al. [2] have observed that research papers combating anti-forensic techniques are vastly outnumbered by the number of websites that discuss how to exploit the digital forensic process [8]. Combating anti-forensics requires a consensus view and a standardized definition

and categories of anti-forensic methods in order to develop mitigation strategies [8]. Definitions of anti-forensics were proposed in 2005 by Rogers [13] and in 2006 by Harris [8]. The most recent definition was formulated in 2016 by Conlan et al. [4], who summarized the previous definitions and defined anti-forensics as "attempts to alter, disrupt, negate, or in any way interfere with scientifically valid forensic investigations."

Anti-forensic techniques fall into several categories. Rogers [13] has proposed four categories: (i) data hiding; (ii) artifact wiping; (iii) trail obfuscation; and (iv) attacks against the digital forensic process and tools. A more recent taxonomy was specified by Conlan et al. [4]. This taxonomy adds a new category to Rogers' classification: (v) possible indications of anti-digital-forensic activity. It also specifies several subcategories in order to create a more comprehensive and up-to-date taxonomy. The methods considered in this research can be classified as data hiding techniques that are mapped to the filesystem manipulation subcategory in the extended taxonomy of Conlan et al. [4].

In general, there are three ways to hide data in an ext filesystem. First, data can be hidden in slack space. This is because ext has a fixed block size and a write operation that (in most cases) does not need an exact multiple of the block size, making slack space available in several places. Of course, it is important to distinguish between different types of slack space, for example, classical file/directory slack space and metadata slack space associated with several filesystem data structures.

The second way is to hide data in reserved space. Multiple locations distributed across the filesystem are reserved for future use, for example, reserved group descriptor growth blocks for future filesystem expansion. Reserved areas can also be leftovers from earlier versions that are no longer used.

The third way is to misuse filesystem structures to hide data. This approach is effective because it is difficult to distinguish hidden data from normal content. An analysis tool will not report an inconsistency if the hidden data matches the normal internal structures (i.e., there is no anomaly). An example is data hidden in inode timestamps, which the `e2fsck` tool would interpret as ordinary timestamps.

## 2.2 Ext4 Filesystem Layout

Figure 1 shows the ext4 filesystem layout. The filesystem allocates disk space in units of blocks comprising multiple sectors; the typical block size is 4 KiB. Blocks are, in turn, grouped into larger units called block groups.

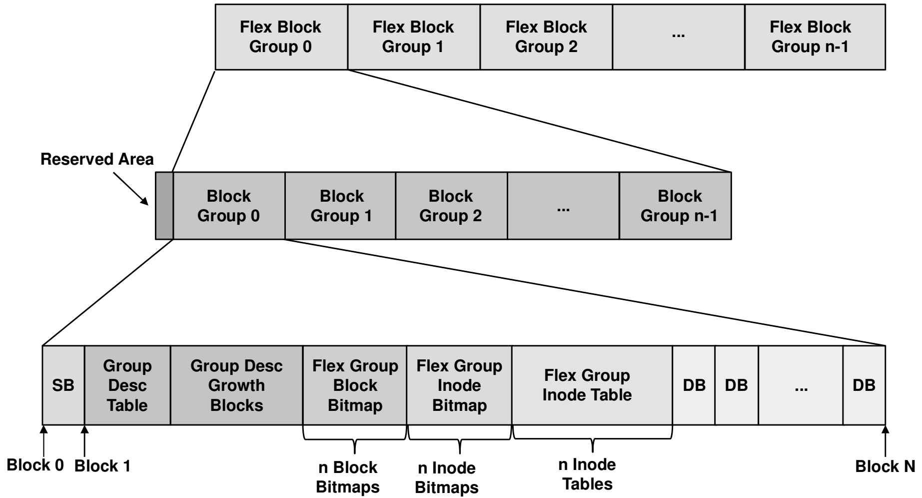



*Figure 1.* General ext4 filesystem layout.

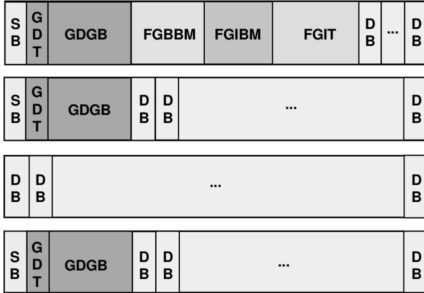
The entire partition is divided into a series of block groups. The ext4 filesystem offers a new feature called a flexible block group (`INCOMPAT_FLEX_BG`). This feature provides better performance by allowing the combination of several block groups into a single logical block group. The block metadata of multiple block groups (i.e., block bitmaps, inode bitmaps and inode tables) are placed close together as one long run in the first block group of the flexible block group.

Figure 2 shows an example ext4 filesystem layout with one flexible block group that includes four block groups (left-hand side of the figure), and the sparse feature.

A block group contains a number of data structures [19]. The first 1,024 bytes are reserved for the boot sector. Following this is the superblock, which generally starts after the reserved area at byte offset 1,024; the superblock is essential to smooth filesystem operation. It records various information about the layout, size and enabled features of the filesystem.

The superblock is followed by the group descriptor table (GDT). Each block group of the filesystem has a table entry, the so-called block group descriptor. It contains metadata about the block group, for example, the locations of the associated block bitmap, inode bitmap and inode table.

To support future expansion of the filesystem, `mke2fs` allocates several group descriptor growth blocks after the group descriptor table. If the sparse feature flag (`RO_COMPAT_SPARSE_SUPER`) is set, then redundant




| Block Grp. | Block Number | Data Structure |
|---|---|---|
| 0 | 0 | Super Block Copy |
| 0 | 1 | Group Descriptor Table |
| 0 | 2 - 62 | Group Descriptor Growth Blocks |
| 0 | 63 | Flex Group Block Bitmap |
| 1 | 63 | Flex Group Block Bitmap |
| 2 | 63 | Flex Group Block Bitmap |
| 3 | 63 | Flex Group Block Bitmap |
| 0 | 67 | Flex Group Inode Bitmap |
| 1 | 68 | Flex Group Inode Bitmap |
| 2 | 69 | Flex Group Inode Bitmap |
| 3 | 70 | Flex Group Inode Bitmap |
| 0 | 71 - 1047 | Flex Group Inode Table |
| 1 | 1048 - 2024 | Flex Group Inode Table |
| 2 | 2025 - 3001 | Flex Group Inode Table |
| 3 | 3002 - 3978 | Flex Group Inode Table |
| 0 | 3979 - 32767 | Data Blocks |
| 1 | 32768 | Super Block Copy |
| 1 | 32769 | Group Descriptor Table |
| 1 | 32770 - 32830 | Group Descriptor Growth Blocks |
| 1 | 32831 - 65535 | Data Blocks |
| 2 | 65536 - 98303 | Data Blocks |
| 3 | 98304 | Super Block Copy |
| 3 | 98305 | Group Descriptor Table |
| 3 | 98306 - 98366 | Group Descriptor Growth Blocks |
| 3 | 98367 - 124999 | Data Blocks |

*Figure 2.* Example ext4 filesystem layout.

copies of the superblock, group descriptor table and group descriptor growth blocks are placed in groups whose group number is 0 or a power of 3, 5 or 7. Otherwise, each block group contains a backup copy.

Following this are the block bitmap, inode bitmap and inode table. The block bitmap tracks the usage of each data block in the block group. The inode bitmap records the entries in the inode table that are in use. The inode table contains the metadata of a file or directory (e.g., file owner, permissions and timestamps). While the bitmaps usually use one block each, the inode table uses multiple contiguous blocks. Leftover space is used by the data blocks that store actual user data.

## 3. Evaluation Methodology

An anti-forensic hiding approach seeks to conceal data. A user who employs such a technique is mainly interested in two aspects. The first aspect is the amount of data that can be hidden by the technique – the more, the merrier. The second is the ease with which hidden data can be discovered by a digital forensic expert – the harder, the better.

*Table 2.* Relevant metadata information in the ext4 test volumes.

| Metadata Information | 64 GiB Volume | 500 GiB Volume |
|---|---|---|
| Block Size | 4,096 | 4,096 |
| Inode Size | 256 | 256 |
| Inode Range | 1 – 4,194,305 | 1 – 32,768,001 |
| Free Blocks | 16,369,541 | 128,734,265 |
| Free Inodes | 4,194,293 | 32,767,989 |
| GDT Size (in Blocks) | 8 | 63 |
| Number of Block Groups | 512 | 4,000 |
| Blocks per Block Group | 32,768 | 32,768 |
| Inodes per Block Group | 8,192 | 8,192 |
| Superblocks (with Sparse Feature) | 13 | 18 |
| Superblocks (without Sparse Feature) | 512 | 4,000 |

Therefore, a hiding approach is evaluated based on: (i) capacity; and (ii) detection rating. Comparable evaluation metrics have not been proposed in the digital forensics literature.

- **Capacity:** The maximum hideable amount of data in bytes over the entire filesystem is specified for each technique. If there is insufficient space for an entire file to be hidden in one location, it can be divided into several pieces and hidden in multiple locations across the filesystem. Furthermore, the maximum hiding capacities of a 64 GiB volume (e.g., Android internal flash memory) and a 500 GiB volume (e.g., laptop SSD) are specified. Reasonable estimates of the hiding capacity were made based on information extracted from sample test images using Sleuthkit (Table 2).

- **Detection Rating:** The following detection rating scheme is employed:

  - The "easy" rating implies that a digital forensic investigator will typically find the hidden data.
  - The "advanced" rating implies that the hidden data would not ordinarily be found, but a digital forensic expert would find it if there is an anomaly.
  - The "difficult" rating implies that the hidden data would hardly ever be found.

The Sleuthkit, FTK Imager and Autopsy tools were used to obtain ratings of the data hiding techniques. Modern volumes are getting larger and forensic investigators often do not have the expertise or resources to perform manual examinations using hex editors. Forensic tools are used

to examine volumes and alert investigators to anomalies. If an alert does not point an investigator to the hidden data, it can remain undetected.

The `e2fsck` and `debugfs` filesystem checking utilities in the `e2fsprogs` suite may be used to find filesystem inconsistencies and provide hints about hidden data. The `e2fsck` utility can be forced to check a filesystem using the `-f` argument. Beyond this, file carving can be used to obtain hidden data, but this is outside the scope of this research, which focuses on data hiding and detection using filesystem structures instead of treating the entire volume as an image without metadata.

It is important to note that detecting hidden data does not solve the problem of interpreting the hidden data. Therefore, after an anomaly indicates the presence of hidden data, reverse engineering and decryption techniques may have to be applied to determine the meaning of the hidden content. Obfuscated and encrypted data significantly complicate hidden data detection using carving techniques.

This research focuses on the capabilities of forensic tools to find hidden data in unexpected locations. When assessing the hiding techniques in this research, it was assumed that a forensic investigator would know the tools well, but not the details about the ext4 filesystem. Furthermore, it was assumed that the investigator would not examine the entire volume manually if nothing unexpected was encountered.

Experiments were performed on several images created using the `dd` command and formatted with certain ext4 settings using `mke2fs`. The data, which included JPG, TXT and ZIP files, and single ASCII characters, was subsequently hidden in the created images using `dd`.

## 4. Hiding Methods Based on Previous Research

This section evaluates whether or not the hiding techniques proposed for ext2/ext3 or other filesystems work on the ext4 filesystem.

The experiments demonstrate that nine of the ten tested techniques still work. Table 3 lists each of the tested methods along with its hiding capacity and detection rating.

### 4.1 File and Directory Slack Space

Data can be hidden in file slack space [5] as well as in directory slack space [3]. Slack space hiding methods developed for previous ext versions also work on the ext4 filesystem.

The available slack space depends on the block size, amount of data stored in the corresponding block and number of allocated files. This makes it possible to conceal $\frac{BlockSize}{2} \cdot UsedInodes$ bytes on average. Just like any other file, ext4 directories are allocated in blocks. They contain

*Table 3.* Summary of adapted ext4 hiding techniques with hiding capacity and detection rating metrics.

| **Data Hiding Technique** | **Capacity (Bytes) (Usable Space)** | **Capacity 64 GiB Volume** | **Capacity 500 GiB Volume** | **Detection Rating** |
|---|---|---|---|---|
| File/Directory Slack Space | $\frac{BlockSize}{2} \cdot UsedInodes$ | 8 GiB | 62.5 GiB | Easy |
| Null Directory Entries | $(BlockSize - (3 \cdot 12) - 8) \cdot UsedDataBlocks$ | 61.77 GiB | 485.8 GiB | Advanced |
| Partition Boot Sector | 1,024 | 1,024 B | 1,024 B | Easy |
| Superblock: Slack Space | $BlockSize - 1,024$ | 1.5 MiB | 11.71 MiB | Easy |
| Superblock: Reserved Space | 394 | – | – | Easy |
| Superblock: Backup Copies | $BlockSize \cdot (BlockGroups - 1)$ | 2 MiB | 15.62 MiB | Advanced |
| Group Descriptor Table: Slack Space | $GDTSize - (GroupDescSize \cdot BlockGroups)$ | – | 7.81 MiB | Easy |
| Block Group Descriptor: Reserved Space | $4 \cdot BlockGroups$ | 2 KiB | 15.62 KiB | Advanced |
| Inode Table: Reserved Inodes | $2 \cdot (InodeSize - 20)$ | 472 B | 472 B | Advanced |
| Inode: Reserved Space | $2 \cdot (Inodes - 8)$ | 8 MiB | 62.5 MiB | Advanced |

```
e2fsck 1.43.7 (16-Oct-2017)
Pass 1: Checking inodes, blocks and sizes.
Pass 2: Checking directory structure.
Directory inode 12, block #0: directory passes checks but fails
    the checksum.  Fix<y>? yes.
```

*Figure 3.* Repair of an directory inode checksum using `e2fsck`.

several directory entries, including at least one entry each for the current directory and parent directory. The available space depends on the number of directory entries. Data can be hidden in the space between the last directory entry and before the `ext4_dir_entry_tail` structure because the last existing entry points to this structure at the end of the block [19], leaving the remaining space unused. The downside of using slack space is that modifying the original file or directory often overwrites the hidden data; consequently, this method should be restricted to static files and directories.

Hidden data in file slack space does not cause an `e2fsck` or kernel warning. However, hidden data in directory slack space causes checksum errors. These can be fixed by `e2fsck` without losing the hidden data as shown in Figure 3. According to the information in Table 2, the 64 GiB volume provides about 8 GiB of available slack space on average whereas the 500 GiB volume provides about 62.5 GiB.

It is common knowledge that data can be hidden in file slack space; therefore, it is classified as easy to find. Common forensic tools check the file slack space by default. However, Autopsy does not show the file slack space automatically; directory slack space is visible using the built-in hex viewer. FTK Imager shows the file slack space in the file browser, but the directory slack space is not visible in the same way. The Sleuthkit command `icat -s` can help extract a file, including its slack space. Directory slack space should not be ignored during an investigation. Many seemingly useless small files or empty directories in a volume could indicate hidden data in file slack space and directory slack space. File carving can help distinguish hidden data from normal binary data in file slack space.

## 4.2 Null Directory Entries

The anti-forensic tool KY FS [14] shows how data can be hidden in an ext2 directory entry. In the ext4 filesystem, this can still be done by setting the values of the `inode` and `name_len` attribute of a directory entry to zero. After this is done, the directory entry appears as unused and is not visible in a normal file explorer. The length `rec_len` is set

to the length of the entire block and data is then hidden in the name field of the entry. Note that 12 bytes each for the current directory, parent directory and structure `ext4_dir_entry_tail` at the end of the block including the checksum, as well as 8 bytes for the null directory entry itself, cannot be used to conceal data. With the exception of the above-mentioned entries $(BlockSize - (3 \cdot 12) - 8) \cdot UsedDataBlocks$ bytes remain to hide data.

In the experiments, 61.77 GiB of data could be hidden in the 64 GiB image and 485.8 GiB in the 500 GiB image. Invalid directory inode checksums can also be fixed by `e2fsck` as shown in Figure 3.

This is an advanced data hiding technique because null directory entries can be difficult to recognize among the variety of directory entries; in fact, their content would appear almost invisible at first sight. Hidden data can be viewed manually using the FTK Imager and Autopsy hex viewers, but no user notifications are provided. Forensic investigators should pay attention to the presence of numerous empty directories.

## 4.3 Partition Boot Sector

Piper et al. [12] have shown that the first 1,024 bytes (boot sector) of an ext2 volume can be used to hide data. This method is applicable to ext4. No errors are detected during a forced filesystem check.

The ability to hide data in the boot sector is well known and a forensic investigator should find it easy to find the hidden data. FTK Imager provides the option to examine the boot sector using its hex viewer. However, Autopsy has no option to show this data structure.

## 4.4 Superblock: Slack Space

Berghel et al. [3] mention that unused space exists behind the superblock up to the end of the block in the ext3 filesystem. In the ext4 filesystem, the superblock also has a total size of 1,024 bytes. In general, this provides $BlockSize - 2,048$ bytes of usable space in block group 0 and $BlockSize - 1,024$ bytes in each remaining block group, including a superblock backup copy, since the first superblock has an offset of 1,024 bytes to make room for the additional boot sector.

Hidden data does not affect `e2fsck`. The available space depends on whether the `sparse_super` feature flag is set. In the experiments, the 64 GiB volume provided 38 KiB of usable space with the sparse feature and 1,535 KiB without the feature. The 500 GiB volume provided 53 KiB of usable space with the sparse feature and 11.71 MiB without it.

Hidden data in the slack space of a superblock is easy to find because this area is normally empty. In fact, a forensic investigator should be

```
e2fsck 1.43.7 (16-Oct-2017)
ext2fs_open2: Superblock checksum does not match superblock.
e2fsck: Superblock invalid, trying backup blocks...
```

*Figure 4.* Automated repair of an invalid superblock.

suspicious if it is not empty. FTK Imager provides the option to examine a superblock with its hex viewer. However, Autopsy has no special option to show this data structure and does not provide any warnings.

## 4.5 Superblock: Reserved Space

The anti-forensic tool Data Mule FS has been shown to use 759 bytes in a superblock to hide data [14]. According to the Linux kernel source code [17], the reserved space in the ext4 superblock is 394 bytes (2 bytes in `s_reserved_pad` and 392 bytes in array `s_reserved[98]`). However, this hiding technique is useless. In the experiments, all the attempts at concealing data in the reserved space of the primary superblock failed because `e2fsck` generated invalid checksum errors. Subsequent recovery using the backups of the superblock overwrote the hidden content (Figure 4). However, as discussed in the next section, this behavior does not apply to the backup copies. Due to the warnings, data hidden in the reserved space of a superblock is easily spotted.

## 4.6 Superblock: Backup Copies

Piper et al. [12] have shown that it is possible to hide data in ext3 superblock backups. This data hiding method is still applicable to ext4. In fact, redundant superblocks can be fully used to hide data as long as the first superblock is undamaged. No warnings are issued when the volume is mounted or checked with `e2fsck`. If the sparse feature flag is not set, then $BlockSize \cdot (BlockGroups - 1)$ bytes are available to conceal data; otherwise, the space available is reduced to the number of block groups including the backups.

In the experiments, the 64 GiB volume provided about 48 KiB of space for data hiding with the sparse feature and 2 MiB without it. The 500 GiB volume provided about 68 KiB with the sparse feature and 15.62 MiB without it. However, data loss could occur if the first superblock is corrupted; this is because `e2fsck` attempts to restore the original data structure in all backups and overwrites the hidden content. Also, if no valid backup is found, a successful filesystem recovery is impossible.

This method is classified as advanced in terms of its difficulty. This is because real data is actually expected in the superblock backups, unlike slack space, which is normally empty. The superblocks can be viewed manually using FTK Imager with its hex viewer, but Autopsy has no such option. The `e2fsck` utility does not point to the manipulated backups, so hidden data may remain undetected. A forensic investigator should check for a missing sparse feature flag, which is set by default. The flag could have been deliberately removed by an attacker to obtain additional space for data hiding.

## 4.7 Group Descriptor Table: Slack Space

Berghel et al. [3] have identified the presence of group descriptor slack space in previous ext versions. The ext4 filesystem still has some slack space behind the group descriptor table when its last block is not completely filled with table entries. Hiding data in this slack space assumes that the filesystem will not grow in size because any added group descriptors would overwrite the hidden data.

The ext4 filesystem extends the group descriptor size from 32 to 64 bytes when the 64-bit feature is enabled. This provides $GDTSize - (GroupDescSize \cdot BlockGroups)$ bytes per table for hiding data. The number of group descriptor table backups depends on the `sparse_super` feature flag. Hidden data does not affect `e2fsck`.

In the experiments, data could not be concealed in the 64 GiB image because all the group descriptor table blocks were completely filled with group descriptors. However, the 500 GiB volume provided 2,048 bytes per table, corresponding to 36 KiB with the sparse feature and 7.81 MiB without the feature for the entire filesystem.

Just like any other method that hides data in slack space, this technique is rated as easy to detect. FTK Imager enables the group descriptor table to be examined using its hex viewer; Autopsy does not provide this option.

## 4.8 Block Group Descriptor: Reserved Space

Data Mule FS has utilized 14 bytes of reserved space to hide data in each ext2 group descriptor [14]. The Linux kernel source code [17] shows that ext4 group descriptors still have a 4-byte `bg_reserved` structure for padding when the 64-bit feature is enabled. The available storage per group descriptor table, which depends on the number of block groups, amounts to $4 \cdot BlockGroups$ bytes. Therefore, each group descriptor table in the 64 GiB volume provides 2 KiB of usable space while each group descriptor table in the 500 GiB volume provides 15.62 KiB of usable space.

```
e2fsck 1.43.7 (16-Oct-2017)
One or more block group descriptor checksums are invalid.
    Fix<y>? yes.
Group descriptor 0 checksum is 0x5009, should be 0x7d85.
    FIXED.
```

*Figure 5.* Repair of an invalid group descriptor checksum.

After manipulating the group descriptors, `e2fsck` can be used to fix invalid checksums (Figure 5). No more errors occur in further filesystem checks.

FTK Imager provides the option to examine the group descriptor table with its hex viewer, but does not give any warnings; Autopsy has no such option. However, due to the fact that only four bytes per group descriptor are available, hidden data would almost certainly be spread over many group descriptors. If a forensic tool does not explicitly report a non-empty reserved field, the fragmented data can render the forensic investigation more difficult. Therefore, this data hiding technique is classified as advanced.

## 4.9 Inode Table: Reserved Inodes

Data hiding using reserved inodes has been discussed for ext2 and ext3 [5, 12]. In ext4, inodes 1 to 10 are reserved for internal filesystem use. Inode 0 is not used. Inode 11 is the first available inode and is typically used for the `lost+found` directory. However, the kernel source code [17] does not explicitly mention the use of inodes 9 and 10. According to Holen [9], inode 9 is used for snapshots by `Next3fs` and inode 10 is used for ext4 metadata replication. Both are non-standard options and are not used without a patched kernel.

```
e2fsck 1.43.7 (16-Oct-2017)
Pass 1: Checking inodes, blocks and sizes.
Inode 12 passes checks, but checksum does not match inode.
    Fix<y>? yes.
```

*Figure 6.* Repair of an invalid inode checksum.

Tests have shown that the fields `i_mode`, `i_blocks_lo`, `l_i_blocks_hi`, `i_flags`, `i_size_high`, `l_i_checksum_lo` and `i_checksum_hi` should not be manipulated because they cause errors in `e2fsck`. Therefore, $2 \cdot (InodeSize - 20)$ bytes remain, which provides 472 bytes in each test volume. The `e2fsck` utility corrects the inode checksums after manipulation (Figure 6).

It should be mentioned that data can be hidden in blocks that are marked bad using the reserved inode 1. The filesystem prevents bad blocks from being allocated to a file or directory. Any block can be added to the list of bad blocks (using `e2fsck [-l bad_blocks_file] device`), which provides almost unlimited space.

FTK Imager supports the examination of the inode table with its hex viewer; Autopsy has no such option. This data hiding technique is classified as advanced because reserved inodes are usually not assumed to be empty. Therefore, all inodes, including reserved inodes and slack space, should be analyzed by default using forensic software.

## 4.10 Inode: Reserved Space

Data Mule FS has utilized 10 bytes per ext2 inode to hide data [14]. However, the ext4 inode structure only has two bytes available in `l_i_reserved` [17]. With the exception of the reserved inodes 1 to 8, this field can be abused, providing $2 \cdot (Inodes - 8)$ bytes for hiding data. Therefore, the 64 GiB volume offers approximately 8 MiB of space for hiding data and the 500 GiB volume offers 62.5 MiB. Invalid inode checksums can be fixed using `e2fsck` after data has been hidden (Figure 6).

FTK Imager provides the option to investigate the inode table manually; Autopsy has no special option.

The inode reserved space is mentioned in the ext4 documentation [19]. Nevertheless, this data hiding method is rated as advanced because the same situation applies as in the case of the group descriptor reserved space. Hidden data is hard to find when it is distributed in multiple entries across the inode table. An investigator should, therefore, keep an eye on the various reserved areas.

# 5. New Hiding Methods

This section discusses locations for hiding data that have been identified during this research project. Some techniques have not been described in the literature because they are only applicable to ext4. Furthermore, this section verifies whether or not any of the potential ext4 hiding places proposed in the literature, but that have not been tested, actually work.

The experiments demonstrate that all eight tested techniques work. Table 4 summarizes the results of the eight techniques.

## 5.1 Block Bitmap: Slack Space

The block bitmap is one block in size because the number of blocks per block group corresponds to the number of bits of one block (e.g.,

*Table 4.* Summary of new ext4 hiding techniques with hiding capacity and detection rating metrics.

| **Data Hiding Technique** | **Capacity (Bytes) (Usable Space)** | **Capacity 64 GiB Volume** | **Capacity 500 GiB Volume** | **Detection Rating** |
|---|---|---|---|---|
| Block Bitmap: Slack Space | $(BlockSize - (\frac{BlocksPerGroup}{8})) \cdot BlockGroups$ | Variable | Variable | Easy |
| Inode Bitmap: Slack Space | $(BlockSize - (\frac{InodesPerGroup}{8})) \cdot BlockGroups$ | 1.5 MiB | 11.71 MiB | Easy |
| Inode: Slack Space/ Extended Attributes | $(InodeSize - (128 + i_extra_isize)) \cdot Inodes$ | 384 MiB | 3,000 MiB | Advanced |
| Inode: Nanosecond Timestamps | $15 \cdot UsedInodes$ | 60 MiB | 469 MiB | Difficult |
| Group Descriptor Table: Backup Copies | $GDTSize \cdot (BlockGroups - 1)$ $GDTSize = BlockSize \cdot \lceil \frac{BlockGroups \cdot s_desc_size}{BlockSize} \rceil$ | 16 MiB | 984 MiB | Advanced |
| Group Descriptor Table: Growth Blocks | $s_reserved_gdt_blocks \cdot BlockSize \cdot (BlockGroups - 1)$ | 2 GiB | 15.6 GiB | Advanced |
| Extent: Persistent Preallocation | 16 TiB | Variable | Variable | Advanced |
| Data Structures in Uninitialized Block Groups | $DataStructureSize \cdot UninitBlockGroups$ | Variable | Variable | Advanced |

there are 32,768 blocks per block group with a default block size of 4,096 bytes). Default settings leave no room between the block bitmap and the end of the block. However, slack space can still exist because the number of blocks per group does not necessarily have to match the number of bits in a block, and it is adjustable during formatting. The command `mkfs.ext4 -g [image]` deliberately creates a filesystem with fewer blocks per group.

Several regions of block bitmap slack space exist, one in each block group. The amount of data that can be hidden depends on the number of block groups and amounts to $(BlockSize - (\frac{BlocksPerGroup}{8})) \cdot BlockGroups$ bytes. The amount of usable slack space varies, but corresponds to the inode bitmap. Hidden data survives a forced check by `e2fsck`.

FTK Imager presents the block bitmaps in its hex viewer; Autopsy has no such option. Hidden data should be easy to find because, with the default filesystem settings, no slack space is available behind the block bitmap; if there is space, it should contain only zeros.

## 5.2 Inode Bitmap: Slack Space

There are fewer inodes than blocks per block group. Therefore, unlike the block bitmap, several bytes remain unused at the end of each inode bitmap with the default settings. The amount of usable space is repeated per block group and this provides $(BlockSize - (\frac{InodesPerGroup}{8})) \cdot BlockGroups$ bytes. The 64 GiB volume can store 1.5 MiB of hidden data and the 500 GiB volume can store 11.71 MiB. The hidden data survives a forced check by `e2fsck`.

FTK Imager provides the option to view the inode bitmaps manually whereas Autopsy does not provide this option. The slack area behind the inode bitmap up to the end of the block normally contains zeros. Any other data would be easily found by a forensic investigator.

## 5.3 Inode: Slack Space/Extended Attributes

The ext2 and ext3 filesystems have an inode size of 128 bytes and leave no space for data hiding. The default size of an inode record in ext4 is 256 bytes [10]. It can even be set to the filesystem block size using the `mkfs.ext4 [-I inode size]` option at format time. The extra 128 bytes are divided into a range of fixed fields and a range of extended attributes as shown in Figure 7. Each inode contains the field `i_extra_isize`, which records the additional number of bytes for the fixed fields beyond the original 128 bytes. The extra space between the end of the inode structure and the end of the inode record is meant to store extended

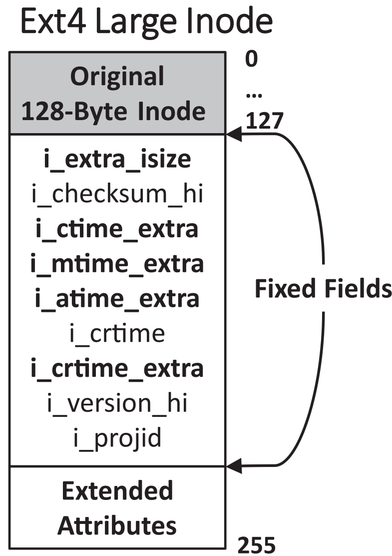



*Figure 7.* Ext4 inode structure layout (adapted from [10]).

attributes, but it can also be used to hide data. The maximum concealable amount of data is $(InodeSize - (128 + i_extra_isize)) \cdot Inodes$. Thus, 384 MiB of space is available in the 64 GiB volume and 3,000 MiB in the 500 GiB volume (when `i_extra_isize` is set to 32). After an inode is manipulated, its checksum can be fixed by `e2fsck` (Figure 6). No warnings occur during additional checks.

If no extended attributes are used, the area behind the actual inode structure is normally empty and any hidden data would be suspicious. Since this is not generally the case and none of the tools give any indications of hidden data, the detection rating is advanced. Any inodes that do not have the default size of 256 bytes should raise further suspicion. In extreme cases, each inode can have the size of an entire block, thereby providing additional space.

## 5.4 Inode: Nanosecond Timestamps

With larger inodes (256 bytes), there is room to support nanosecond timestamps, so additional 32-bit `i_[c|m|a|cr]time_extra` fields were added to the original inode structure as shown in Figure 7. Since 30 bits are sufficient to enable nanosecond precision, the remaining two bits are used to extend the Unix epoch (new overflow date is `2446-05-10`) [19]. If the additional precision for the timestamps is not required, then the

four fields can conceal 16 bytes in each inode. However, the use of the lower two epoch bits leads to dates beyond the year 2038, which looks suspicious and could help reveal the hidden data. Therefore, it makes more sense to hide data only in the upper 30 bits of the nanosecond timestamps. This provides $15 \cdot \mathit{UsedInodes}$ bytes if all four timestamps are used. Thus, the 64 GiB volume offers almost 60 MiB of usable space while the 500GiB volume offers as much as 469 MiB of space. After the checksums of the manipulated inode entries are repaired (Figure 6), hidden data survives another forced `e2fsck` check and no warnings are given when the filesystem is mounted.

Data hidden in this manner is difficult to find. Common file explorers, as well as `ls -la`, do not support nanosecond accuracy. FTK Imager and Autopsy also do not show the nanosecond timestamps in their file explorers. Commands such as `stat [file]` or `debugfs -R 'stat <inode>' [image]` can parse the timestamps, but this does not provide concrete information about the hidden data. Furthermore, tests have shown that the `istat` command of Sleuthkit does not take the extra epoch bits into account and, therefore, timestamps beyond 2038 are not decoded properly. A forensic investigator should keep an eye on access or modification timestamps that occur before the creation time.

## 5.5 Group Descriptor Table: Backup Copies

During this research it was discovered that, in addition to the superblock backup copies, backups of the group descriptor table can be used to hide data. The amount of space depends on the size of the group descriptor table and corresponds to $\mathit{GDTSize} \cdot (\mathit{BlockGroups} - 1)$ if the sparse feature is disabled; otherwise, the space is reduced to the number of block groups including the backups. The 64 GiB volume provides 16 MiB and 384 KiB of usable space without and with the feature, respectively; the corresponding values for the 500 GiB volume are 984 MiB and 4 MiB, respectively. The `e2fsck` utility does not give any warnings. However, when the first group descriptor table is damaged, any hidden data is overwritten during the filesystem check.

As in the case of the superblock backups, this method is rated as advanced because real data is typically already present in this location. FTK Imager provides the option to examine the group descriptor table manually, but Autopsy does not. None of the tools give any warnings.

## 5.6 Group Descriptor Table: Growth Blocks

Reserved group descriptor table growth blocks enable the expansion of the group descriptor table and filesystem. Fairbanks [6] points out

```
e2fsck 1.43.7 (16-Oct-2017)
Pass 1: Checking inodes, blocks and sizes.
Inode 7 has illegal block(s). Clear<y>? no.
Too many illegal blocks in inode 7.
```

*Figure 8.* `e2fsck` error after group descriptor table growth block manipulation.

that these additional blocks may be used to hide data. The number of reserved group descriptor table blocks for future filesystem expansion is stored in the superblock if the feature flag `COMPAT_RESIZE_INODE` is set. All attempts to hide data in reserved group descriptor table blocks of block group 0 failed because `e2fsck` generated errors (Figure 8).

However, there are several backups in other block groups where data can be hidden without raising any warnings (as in the case of the superblock/group descriptor table backups). The amount of space corresponds to $s_reserved_gdt_blocks \cdot BlockSize \cdot (BlockGroups - 1)$. The 64 GiB volume provides about 2 GiB of space whereas the 500 GiB volume provides nearly 15.6 GiB of usable space.

Because of the `e2fsck` warning, hidden data in block group 0 is discovered easily. In this case, the error message relates to inode 7 (reserved group descriptor inode), which is an indication of errors in the group descriptor table blocks. However, this does not apply to other block groups. The data hiding technique is rated as advanced because the growth blocks are not a well-known ext4 data structure. FTK Imager and Autopsy do not provide any options to examine this data structure.

## 5.7 Extent: Persistent Preallocation

In the ext4 filesystem, the old indirect block mapping scheme is replaced with an extent tree. An extent no longer provides a one-to-one mapping from logical blocks to disk blocks; instead, it efficiently maps a large part of a file to a range of contiguous physical blocks [10]. Instead of saving many individual block numbers, an extent only has to save the first block number that it covers, the number of blocks it covers and the physical block number to which it points (Figure 9). The persistent preallocation feature permits the preallocation of blocks for a file, which typically extends its size (e.g., database) without having to initialize the blocks with valid data or zeros [10]. The most significant bit of the `ee_len` field in the `ext4_extent` structure indicates whether an extent contains uninitialized data. If the bit is set to one, the filesystem only returns zeros during a read of the uninitialized extent. For this reason, data can be hidden using persistent preallocation. Hidden data does not induce `e2fsck` errors or kernel warnings when a volume is mounted. The

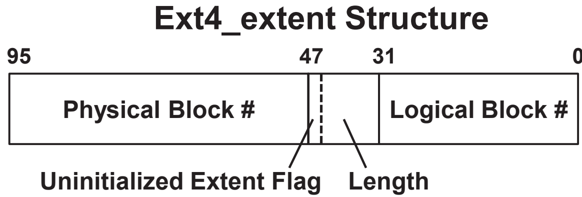



*Figure 9.* Ext4 extent structure (from [10]).

amount of space is limited to the maximum file size, which is 16 TiB in the case of 4 KiB blocks [6].

This is rated as an advanced data hiding technique because the uninitialized extent flag can be overlooked. However, few forensic investigators are knowledgeable about the new features introduced in ext4. Data that returns only zeros should be deemed suspicious. Additionally, Fairbanks [6] points out that uninitialized extents may contain remnants of previous data, making it even more important to examine them. Forensic tools should have the ability to show the real content of preallocated blocks instead of zero-filled blocks.

## 5.8 Uninitialized Block Groups

Data could be stored in several data structures in uninitialized block groups. This technique can be applied as long as the `uninit_bg` flag (`RO_COMPAT_GDT_CSUM`) is set on the volume. Three block group descriptor flags enable `mke2fs` to skip the initialization of parts of the block group metadata. The `INODE_UNINIT` and `BLOCK_UNINIT` flags enable the inode table/bitmap and block bitmap for the block group to be calculated and, therefore, the on-disk bitmap and table blocks are not initialized immediately during formatting. This is generally the case for an empty block group that only contains fixed-location metadata.

This technique provides considerable storage for hidden data. In fact, the entire space of the uninitialized inode/block bitmaps or inode tables can be used to hide data, instead of just the slack space. It is better to use block groups with high group numbers because they are initialized later. This offers $DataStructureSize \cdot UninitBlockGroups$ bytes for concealing data.

Hidden data does not affect a forced check by `e2fsck`. FTK Imager enables inode bitmaps to be viewed manually using its hex viewer; Autopsy has no such option. This technique has an advanced detection rating because many forensic investigators are not aware of this new ext4 data structure.

## 6. Conclusions

Hidden data in ext4 filesystems may constitute valuable evidence in forensic investigations and should not be underestimated. Seventeen of the eighteen data hiding techniques tested in this research were found to be successful. New hiding places were discovered, previously-proposed techniques were proven to work and even very old methods were verified as still applicable. Most of the data hiding techniques exploit unallocated block space or specific reserved metadata fields of the ext4 filesystem.

The usable data hiding capacity strongly depends on the data structure and the filesystem settings and its size. Ext4 volumes are easily set up to provide enough space to hide data; for example, the sparse feature can be disabled, inode size can be set to the size of an entire block and larger numbers of inodes than usual can be created. The use of non-standard filesystem settings should be cause for alarm in a digital forensic investigation. However, even with default settings, adequate space – ranging from a few bytes to several gigabytes – is available to conceal data in the filesystem data structures. For example, malware could be hidden in various locations (e.g., on an Android smartphone) to remain partially undetectable. Also, data can be hidden in several data structures and distributed across the filesystem, making a forensic investigation much more difficult.

This research has shown that existing forensic tools, as well as filesystem checking utilities, do not recognize hidden data in locations that are not normally used. Ten of the techniques tested are rated as advanced and, therefore, require substantial forensic expertise. Data hidden in nanosecond timestamps is difficult to detect because it would be spread over several timestamps and would be very similar to normal data. The detection rates of forensic tools could be improved if official filesystem specifications were published, including standards of how reserved fields and unused space should be treated. This would help forensic tools keep up with changes to the specifications, enabling them to interpret all the data structures correctly and provide automatic warnings when hidden data is discovered.

Future research should focus on unresearched locations such as hash tree directories and non-essential fields such as the unused field in the extent tree. The use of the ext4 journal for data hiding should also investigated. Another important topic is the measurement of entropy of data in potential hiding places, which would enhance the detection of hidden data because anomalies are easily discovered in locations that normally contain null bytes. Finally, the implementation of an anti-

forensic data hiding toolkit and detection utility would be invaluable to the digital forensic community.

## Acknowledgement

This research was supported by the German Federal Ministry of Education and Research (BMBF) under the funding program Forschung an Fachhochschulen (Contract No. 13FH019IB6) and by the Hessen State (Germany) Ministry for Higher Education, Research and the Arts (HMWK) under CRISP (`www.crisp-da.de`).

## References

[1] R. Anderson, R. Needham and A. Shamir, The steganographic file system, *Proceedings of the Second International Workshop on Information Hiding*, pp. 73–82, 1998.

[2] I. Baggili, A. BaAbdallah, D. Al-Safi and A. Marrington, Research trends in digital forensic science: An empirical analysis of published research, *Proceedings of the Fourth International Conference on Digital Forensics and Cyber Crime*, pp. 144–157, 2012.

[3] H. Berghel, D. Hoelzer and M. Sthultz, Data hiding tactics for Windows and Unix file systems, *Advances in Computers*, vol. 74, pp. 1–17, 2008.

[4] K. Conlan, I. Baggili and F. Breitinger, Anti-forensics: Furthering digital forensic science through a new extended granular taxonomy, *Digital Investigation*, vol. 18(S), pp. S66–S75, 2016.

[5] K. Eckstein and M. Jahnke, Data hiding in journaling file systems, *Proceedings of the Fifth Digital Forensic Research Workshop*, 2005.

[6] K. Fairbanks, An analysis of Ext4 for digital forensics, *Digital Investigation*, vol. 9(S), pp. S118–S130, 2012.

[7] S. Garfinkel, Anti-forensics: Techniques, detection and countermeasures, *Proceedings of the Second International Conference on Information Warfare and Security*, pp. 77–84, 2007.

[8] R. Harris, Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem, *Digital Investigation*, vol. 3(S), pp. S44–S49, 2006.

[9] V. Holen, Reserved ext2/ext3/ext4 inodes (`www.vidarholen.net/contents/junk/inodes.html`), 2012.

[10] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas and L. Vivier, The new Ext4 filesystem: Current status and future plans, *Proceedings of the Linux Symposium*, vol. 2, pp. 21–33, 2007.

[11] A. McDonald and M. Kuhn, StegFS: A steganographic file system for Linux, *Proceedings of the Third International Workshop on Information Hiding*, pp. 463–477, 1999.

[12] S. Piper, M. Davis, G. Manes and S. Shenoi, Detecting hidden data in Ext2/Ext3 file systems, in *Advances in Digital Forensics*, M. Pollitt and S. Shenoi (Eds.), Springer, Boston, Massachusetts, pp. 245–256, 2005.

[13] M. Rogers, Anti-forensics, presented at Lockheed Martin, San Diego, California, September 15, 2005.

[14] The Grugq, The art of defiling: Defeating forensic analysis, presented at *Black Hat USA*, 2005.

[15] C. Thuen, Understanding Counter-Forensics to Ensure a Successful Investigation, Department of Computer Science, University of Idaho, Moscow, Idaho (pdfs.semanticscholar.org/d5b6/b658d9178dbcdf33e095a53c45b4f7a43fc8.pdf), 2007.

[16] T. Ts'o, Android will be using ext4 starting with Gingerbread, Blog Entry (thunk.org/tytso/blog/2010/12/12/android-will-be-using-ext4-starting-with-gingerbread), December 12, 2010.

[17] T. Ts'o, Ext4 filesystem tree, Kernel.org git repositories (git.kernel.org/pub/scm/linux/kernel/git/tytso/ext4.git), 2018.

[18] D. Wong, Ext4 Howto, *Ext4 Wiki* (ext4.wiki.kernel.org/index.php/Ext4_Howto), 2015.

[19] D. Wong, Ext4 Disk Layout, *Ext4 Wiki* (ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout), 2016.