

available at www.sciencedirect.comjournal homepage: www.elsevier.com/locate/diin
**Digital
Investigation**

Contagious errors: Understanding and avoiding issues with imaging drives containing faulty sectors[☆]

David Byers*, Nahid Shahmehri

Department of Computer and Information Science, Linköping University, SE-58183 Linköping, Sweden

ARTICLE INFO

Article history:

Received 23 January 2008

Received in revised form 13 March 2008

Accepted 17 March 2008

Keywords:

Acquisition of digital data

Hard drive imaging

Testing forensic tools

Faulty sectors

IDE/ATA Linux

DCFLdd

dd

sdd

ddrescue

dd_rescue

ABSTRACT

When using certain tools to image drives that contain faulty sectors, the tool may fail to acquire a run of sectors even though only one of the sectors is really faulty. This phenomenon, which we have dubbed “contagious errors” was reported by James Lyle and Mark Wozar in a recent paper presented at DFRWS 2007 [Lyle, J., Wozar, M. Issues with imaging drives containing faulty sectors. *Digital Investigation* 2007;4S: S13–5.]. Their results agree with our own experience from testing disk imaging software as part of our work for the Swedish National Laboratory of Forensic Science.

We have explored the issue further, in order to determine the cause of contagious errors and to find ways around the issue. In this paper we present our analysis of the cause of contagious errors as well as several ways practitioners can avoid the problem. In addition we present our insights into the problem of consistently faulty drives in forensic tool testing.

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

When imaging a drive that contains faulty sectors, it is important that all good sectors are correctly imaged even if they happen to be located in close proximity to faulty sectors. In theory this should not be a problem for any tool that reads one sector at a time from the drive. In practice, however, many such tools will fail to read a run of sectors even though only one of them is really faulty.

Lyle and Wozar reported on this phenomenon in a recent DFRWS paper (Lyle and Wozar, 2007), but did not determine the cause of the problem or propose remedies. Our own experiences with testing disk imaging tools confirm the

observations of Lyle and Wozar. We have analyzed the issue further and can offer an explanation, workarounds for practitioners, as well as recommendations for tool developers.

The security and networks group at the Department of Computer and Information Science at Linköping University is currently engaged in a project for the Swedish National Laboratory of Forensic Science which involves (among other activities) conducting comprehensive tests of disk imaging tools using a methodology similar to that developed by NIST in the Computer Forensics Tool Testing (CFTT) project (NIST, 2007). One of the issues being investigated is how the tested tools handle drives with faulty sectors. During the initial phases of this project we noticed that, on Linux, single faulty

[☆] This work was supported by the Swedish National Laboratory of Forensic Science.

* Corresponding author. Tel.: +46 13 282821.

E-mail addresses: davby@ida.liu.se (D. Byers), nahsh@ida.liu.se (N. Shahmehri).

1742-2876/\$ – see front matter © 2008 Elsevier Ltd. All rights reserved.

doi:10.1016/j.diin.2008.03.002

sectors would result in multiple sectors not being read from the drive. The problem was observed with nearly all tools we tested. We have termed this problem “contagious errors”.

In this paper we present the following results from our project:

- Experience with testing faulty drives, and the problem of identifying or creating reliably faulty drives for testing. This information should be of interest to anyone evaluating disk imaging tools.
- Systematic tests of the contagious error problem, used to determine the nature of the problem and which tools are affected by it, as well as an analysis of the problem to determine its cause.
- Several workarounds that can be applied by practitioners to avoid the problem of contagious errors.
- Recommendations for practitioners and tool developers related to this and other similar issues.

2. Reliably faulty drives

One of the greatest challenges when testing drive imaging software is testing imaging of faulty drives. Since tests should always be reproducible, we need reliably faulty drives – drives that are always faulty at the same locations. In the following section we consider only faults when reading; faults when writing are not as great a concern in testing.

Although it is generally very easy to find a drive that exhibits read errors, we have experienced several issues that led us to avoid drives with real faults in testing:

- Most read errors are not caused by media errors, but by ECC errors on the drive (i.e. there are more bit errors in a single sector than the error correcting code the drive uses can cope with). Writing to a sector with this kind of error clears the error, at least for a time. Out of 12 drives that reported read errors we tested, only one had hard media errors; the rest reported zero errors after having been wiped.
- Modern drives automatically remap faulty sectors. When a drive determines that a particular sector is bad, it will place it in a queue to be remapped. On the next write to that sector, the drive will automatically remap it to a good sector taken from a small pool of spare sectors. Because of this, a drive will not reliably exhibit faults until the entire pool of spare sectors has been exhausted, and at this point the drive is often too bad to use in testing.
- Once a drive starts exhibiting media errors, more errors soon follow, so although the drive remains faulty, it is not reliably faulty as more and more sectors start to go bad.
- Operating systems and disk controllers tend not to like faulty drives. One particular controller we used would, after certain types or an undetermined number of errors, completely refuse to read anything from any drive, good or bad. It would also clear the host computer’s settings, so it would forget which device it should boot from. Obviously such hardware needs to be avoided in testing.

Because of these problems we have essentially abandoned the use of drives with actual media errors in testing. Instead

we have turned to an obsolete feature of the ATA command set that is nevertheless widely supported today. In the future we will be able to use a similar, but entirely supported, feature.

The method we use to create errors is also used by some DOS-based hard disk tools, notably MHDD (Postrigan, 2005).

2.1. WRITE LONG

The ATA (Advanced Technology Attachment) standard defines the most popular interface for storage devices in use today. The first ATA standard was finalized in 1994 (ATA-1, 1994). The current version was released in 2005 (ATA-7, 2005).

ATA-1 through ATA-3 (1997) include a command `WRITE LONG`, which writes a single data sector plus a minimum of four vendor-specific bytes. ATA-4 (1998) (and all subsequent revisions) marks `WRITE LONG` as obsolete, but most drives on the market still support it.

In all drives we have tested the vendor-specific bytes written by `WRITE LONG` are the ECC data for the sector read or written. Normal write commands would compute these bytes automatically. If a sector has incorrect ECC data, then subsequent read commands for that sector will report an error. This means that media errors can be simulated by using `WRITE LONG` to write invalid ECC data to a sector.

This approach has some limitations:

- It can only be used on the first 2^{28} sectors of the drive since the commands do not support 48-bit sector addresses. For testing purposes we have not found this to be limiting in any way.
- It cannot be used to simulate write errors since writing to a sector made faulty through this method will clear the error.
- Not all operating systems support issuing `WRITE LONG` commands. The conventional IDE drivers on Linux, for example, assume that all reads and writes are in multiples of 512 bytes. As a result, Linux will neither write nor read any vendor-specific bytes. Recent versions of Linux support `WRITE LONG` when using libATA drivers (typically used for serial ATA drives).

We have created a tool that is capable of both creating and repairing unreadable sectors on Linux using this technique.

2.2. WRITE UNCORRECTABLE EXT

The next version of the ATA standard (ATA-8) (INCITS Technical Committee T13, 2007) is expected to include a new command, `WRITE UNCORRECTABLE EXT`, that creates unreadable sectors in much the same way as the write long trick does. The most important difference is that `WRITE UNCORRECTABLE EXT` supports 48-bit addresses and will be fully supported and documented in the ATA-8 standard.

At the time of writing, there are a few drives on the market that support `WRITE UNCORRECTABLE EXT`, even though the ATA-8 standard is still in draft. Once ATA-8 compliant drives become widely available, `WRITE UNCORRECTABLE EXT` will become our tool of choice for creating reliably faulty drives.

3. Contagious errors

We have conducted tests on contagious errors using several variants of `dd` (we have conducted tests with a few other tools as well, but the results of those tests are not reported here).

3.1. Methodology

The testing methodology was as follows:

- (1) Write a predictable pattern of data to the drive, so that the resulting disk image can be easily verified. We used the `diskwipe` utility developed by NIST (2007) for this step.
- (2) Write invalid ECC data to a single sector to make the drive faulty. We used `atatool`, our custom ATA drive manipulation tool for this step.
- (3) Reset the drive and operating system drivers. We used the command `hdparm -w /dev/hda` for this step.
- (4) Read the drive using each of the tools under test. The following command lines were used:

```
dd if=/dev/hda of=out.dd bs=512 \
    conv=sync,noerror

dcfldd if=/dev/hda of=out.dcfldd \
    bs=512 conv=sync,noerror

sdd -noerror try=1 if=/dev/hda \
    of=out.sdd bs=512

ddrescue -v -c 1 /dev/hda out.ddr
dd_rescue -v -b 512 /dev/hda out.dd_r
```

- (5) Repair the bad sector. We used `atatool` for this step.
- (6) Determine which sectors were not successfully read by examining the output from each tool, the system logs and/or the acquired image.

The drive used was a Seagate ST3160023A 160 Gb ATA drive with firmware revision 3.06. The operating system was Linux 2.6.22. We conducted all tests using the conventional IDE drivers rather than the newer `libATA` drivers, as this is still the default in many Linux distributions (subsequent tests have verified that the problem of contagious errors is independent of driver selection).

3.2. Results

The results for all tools were identical, and are shown in Table 1. A single unreadable sector resulted in eight sectors not being read. This is consistent with the results presented by Lyle and Wozar.

3.3. Analysis

The consistency of the results is something of an illusion: when conducting similar tests on drives that have been in use for some time, the runs of sectors not acquired are

Table 1 – Runs of sectors not acquired

Bad sector number	Sectors not acquired	Run length
63	56–63	8
64	64–71	8
65	64–71	8
71	64–71	8
72	72–79	8
73	72–79	8
98,759	98,752–98,759	8
98,760	98,760–98,767	8
98,761	98,760–98,767	8

typically much larger. Nevertheless, the fact that all tools exhibit very similar problems is a strong indication that the problem is not with the tools themselves, but with the operating system or the way the tools interact with the operating system.

To further analyze the issue, we need to know how the tools interact with the operating system. The following listing is an excerpt of the system calls made by `dd`; the other programs behave in a similar fashion:

```
open("/dev/hda", O_RDONLY|O_LARGEFILE) = 0
...
read(0, "0000"... , 512) = 512
write(1, "0000"... , 512) = 512
read(0, "0000"... , 512) = 512
write(1, "0000"... , 512) = 512
...
```

This shows that `dd` first opens the device for the drive (`open("/dev/hda", ...)`) read-only (`O_RDONLY`), and with an option to read more than 2^{32} bytes of data (`O_LARGEFILE`). It then goes on to read 512 bytes (a single sector) at a time from the drive.

We examined how the Linux operating system implements the read operation under these conditions. The read call goes through the kernel page cache, and when reading data into the page cache, Linux always reads an entire memory page, which is 4096 bytes on 32-bit Intel processors. In other words, when a program requests a single sector from a drive, Linux will actually read eight sectors, and if any one of those sectors is faulty, it will fail to read all eight. This is the cause of contagious errors.

The page cache can be avoided by using direct I/O mode, in which data are transferred directly to application memory from the device being read. A suitable strategy for reading a device in direct I/O mode is to first read multiple sectors at a time, then re-read any failed sectors in successively smaller increments (e.g. start reading four sectors at a time, and if a read fails, re-read those four sectors two at a time and if one of those reads fails, re-read those two sectors one at a time). This strategy gives both good performance and accuracy (since areas around errors are read one sector at a time). A strategy like this is used by e.g. `ddrescue`.

Several versions of `dd` support direct I/O (`if lag=direct` for regular `dd`; `-d` for `dd_rescue` and `ddrescue`).

4. Overcoming the contagious error issue

Ideally, we would like to be able to disable the page cache or always force direct I/O mode on a specific device, but this does not appear to be possible in Linux. There are, however, other ways of getting around the problem.

4.1. Rewriting the software

Open source software, such as `sdd` can be rewritten to support direct I/O mode. Such a change is generally simple to make. For performance reasons, it would be ideal if normal I/O was used whenever possible, reverting to direct I/O mode only when errors are encountered, but that is a more difficult change to make.

Unfortunately, rewriting the software is not always an option.

4.2. Intercepting the open system call

The behavior of dynamically linked programs can be changed without altering the software. Using the `LD_PRELOAD` feature of Linux, it is possible to intercept the `open` call and add the `O_DIRECT` flag. Listing 1 shows code that accomplishes this. By compiling this code to a shared library and indicating that library in the `LD_PRELOAD` environment variable when running a program, we can force the program to use direct I/O.

```
#define _GNU_SOURCE
#include <dlfcn.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

static int (*next_open)(const char *pathname,
                        int flags, ...);

int open(const char *pathname, int flags, ...) {
    char *msg;

    if (next_open == NULL) {
        next_open = dlsym(RTLD_NEXT, "open");
        if ((msg = dlerror()) != NULL)
            exit(1);
    }

    if ((msg = getenv("DIRECT_DEVICES"))
        && strstr(msg, pathname))
        flags |= O_DIRECT;

    return next_open(pathname, flags, 0);
}
```

Listing 1 – Wrapper for `open` that forces direct I/O.

But not all programs are dynamically linked. Statically linked programs are not affected by this method.

4.3. Using raw devices

Raw devices are special devices on Linux that can be bound to any block device (such as a hard disk) so that requests to the raw device are actually handled by the bound device. I/O through the raw device does not use the page cache, so errors are not contagious.

When using a statically linked program that cannot be modified to image a hard drive, the block device corresponding to the hard drive can be bound to a raw device, and the image acquired from the raw device. This avoids the problem of contagious errors. For example, with `sdd` the following commands can be used to image `/dev/hda`:

```
raw /dev/raw/raw0 /dev/hda
sdd if=/dev/raw/raw0 of=out.sdd \
    bs=512 -noerror -fill
```

Some caveats apply. Firstly, not all programs can successfully read from a raw device because data must be read to properly aligned memory (for example, `dd` was not always capable of reading from raw devices). Secondly, raw I/O may disappear in future versions of Linux.

4.4. Hopeless situations

Sometimes the situation is hopeless. There are several tools on the market that will normally read large blocks of data from the disk. When errors are detected, they start reading successively smaller blocks until all the data except the error have been read. In principle this is a very good design as it does not sacrifice performance or accuracy.

We have tested a number of different tools, including commercial tools, and found that there are a few situations where none of the workarounds presented in this paper work. The only way to be certain of a tool's performance in the presence of errors, and the applicability of these workarounds, is testing.

5. Conclusions

Contagious errors happen when faulty sectors on a drive cause a disk imaging tool to fail to read good sectors. We have determined that the problem is due to the behavior of the operating system, which is tuned for performance, not forensic discovery. We have also offered some workarounds that practitioners can apply today, while waiting for better software.

Contagious errors are just one representative of a larger class of issues that arise from the interaction between applications and the operating system. Operating systems are designed around the needs of general-purpose applications that for the most part need performance more than they need absolute predictability in low-level accesses to I/O devices. It is likely that the community will eventually discover other problems in this general category.

These issues are made apparent in forensic software testing, where we systematically expose software to a variety of technical variations. While the results in this paper appear

very consistent, a different version of the operating system, or a 64-bit, rather than 32-bit, processor might have given us other results.

Our experience clearly shows that a deep understanding of the interactions with the operating system is vital in order to write good disk imaging software. Without such understanding we end up with problems like contagious errors. Early indications from our current project are that contagious errors are not the only problems arising from a lack of understanding of how the software interacts with the operating system.

Practitioners also need to understand how the operating systems and software they use interacts and the features that their tools provide. For example, a practitioner using `dd` today can avoid the problem of contagious errors entirely by specifying direct I/O.

Forensic tool testing programs, like the CFTT and the project we are currently involved in, also have an important role to play. As independent testers we can expose software to technical variations that vendors may have overlooked, and through an extensive program of testing we build understanding of the conditions under which the software operates

optimally, which in turn allows us to make concrete recommendations to improve practice in the field.

REFERENCES

- ATA-1. AT attachment interface for disk drives. Standard ANSI X3.221-1994; 1994.
- ATA-3. AT attachment-3 interface (ATA-3). Standard ANSI X3.298-1997; 1997.
- ATA-4. AT attachment with packet interface extension (ATA/ATAPI-4). Standard ANSI INCITS 317-1998; 1998.
- ATA-7. AT attachment with packet interface – 7 (ATA/ATAPI-7). Standard ANSI INCITS 397-2005; 2005.
- INCITS Technical Committee T13. AT attachment 8-ATA/ATAPI command set (ATA8-ACS). INCITS T13 Working Draft T13/1699-D, revision 4a; May 21, 2007.
- Lyle J, Wozar M. Issues with imaging drives containing faulty sectors. *Digital Investigation* 2007;4S:S13-5.
- NIST. NIST computer forensics tool testing project [accessed May 2007], <http://www.cfft.nist.gov>; 2007.
- Postrigan D. MHDD [accessed December 2007], <http://hddguru.com/content/en/software/2005.10.02-MHDD/>; 2005.