Recovering Data from Windows Systems by Using Linux

Published by the Open Source Software at Microsoft, May 2007
Special thanks to Chris Travers, Contributing Author to the Open Source Software Lab
Most current version will be maintained at: http://port25.technet.com

Introduction

We have all run into cases where Windows refuses to load for one reason or another. The problem may be a hardware or a software failure, and the problem may seem to be irrecoverable. Yet often Linux can be used to help recover data that otherwise might be lost.

Another application of these processes is in the creation of disk images for post-mortem analysis of security breaches. While such images are not created according to forensics standards (which usually requires special hardware) and would not be likely to be of help in legal cases, they are helpful in internal reviews following such incidents.

In writing this paper, I generally assume that either a Linux workstation is available to accept recovery information or that a USB drive of sufficient size is available to hold the data. Generally in either case, enough free space should be available to store the entire hard drive as a bit-for-bit file and still have at least 2GB of space free. However, if more space is available, the process of organizing the recovered data is a bit easier. If neither of these are available, the utility of Linux in data recovery will be limited.

Finally, if things go badly wrong, additional tools (such as The Coroners Toolkit) can be used but they require a great deal more free space to work. Generally in these cases, you should have enough free space to store the hard drive twice on the same drive.

This paper is written with the assumption that the reader is comfortable on the Linux command line, and can perform basic file operations (mv, cp, navigate the filesystem, and the like). In general the command line is preferred for these sorts of tasks because it is easier to adapt the commands to specific needs than it would be with a GUI, and because when data is at stake, complete control over the process is often desirable.

The commands given in this paper are samples only and can easily be adapted to other circumstances. Although I endeavor to provide enough information to make the process to make such modifications clear, complex scenarios may require review of man pages.

A Few Cautions before we Begin

In cases of data recovery, it is generally important to understand the problem so that further damage is not caused in the process. In cases where drive failure is involved, it is almost always safest to send the drive to dedicated data recovery specialists who can disassemble the drive as needed. This process is, however, expensive and must be weighed against the value of the data being recovered.

Even if there are no signs of drive failure, the safest solution is to assume that the drive is failing. This is not a common problem but can be caused by overheating (and the data is more likely to be corrupted by other parts of the system) or by physical damage to key data sectors. However, because this does not add much to the complexity of the recovery and adds a great deal of safety, I recommend avoiding reading the drive more than once and avoid doing any writes.

The instructions in this document are believed to be reasonably safe. However, data recovery is a field where there are many points where mistakes can lead to irrecoverable data loss. If you are unfamiliar with a command or what it does, consult the manual page, and please do not use data you need for learning these techniques. And if you are unsure of your abilities, hire a data recovery specialist if the data is worth the cost of recovery.

Finally, if images are being made for legal purposes, please hire a forensic specialist. There are specific processes which must be followed if one expects evidence to hold up in court. These processes and the hardware devices associated with them are well beyond the scope of this paper.

Making the Image

The first step in performing any data recovery is usually to create an image of each filesystem on the drive. The images can be stored on a USB hard drive, the system hard drive, a network file share, or somewhere else.

By creating an image, we allow ourselves to do two things we could not otherwise do. First we allow ourselves to screw up and still have a chance to get back to where we started. In cases where the filesystem is damaged, this sort of mistake is not uncommon. Secondly, we reduce the stress we put on a potentially failing hard drive so that no more data is lost than absolutely necessary.

The easiest way to handle this sort of data is to make a separate image file for each of the filesystems to be subject to recovery.

Locating the Drive

Linux hard drives are generally found in /dev/ and follow the following naming conventions: IDE drives follow hd (for hard drive) followed by a letter. hda is the primary master, hdb is the primary slave, hdc is the secondary master, hdd is the secondary slave etc.

All other hard drives generally go through the SCSI interface. This includes all SCSI, USB, and SATA drives. In the 2.4 series of the Linux kernel, it was even possible to force an IDE drive to be reported as a SCSI one though this is unlikely to be encountered in a data recovery scenario.

The following list of devices are the most likely disks to have failed:

- Primary IDE Master: /dev/hda
- Primary IDE Slave: /dev/hdb
- Secondary IDE Master: /dev/hdc
- Secondary IDE Slave: /dev/hdd
- First SCSI, SATA, or USB hard drive: /dev/sda
- Second SCSI, SATA, or USB hard drive: /dev/sdb

Using sfdisk

Once a set of drive candidates have been obtained, it is generally a good idea to check them with sfdisk -l. This process is considered safer than using fdisk in part because the commands are simpler and noninteractive. They require root access.

If the failed drive is the primary IDE master drive, use the following command. Otherwise substitute /dev/hda with the device path listed above.

```
bash# /sbin/sfdisk -l /dev/hda
```

The output of this command provides information on the hard drive geometry (heads, sectors and cylinders), and then lists the partition information. Check the geometry and capacity against what is known about the failed drive so that you don't "recover" data from the wrong device.

Partitions must start and end on cylinder boundaries. Unfortunately many older versions of Windows had trouble with disks which had few heads and a large number of cylinders, some BIOS implementations and drives lie about their geometry. Because Linux tends to read the geometry from the drive, and Windows tends to read it from the BIOS, problems can occur when the drive-read an BIOS-read geometries don't match.

Run against a standard Linux disk, this might output something like:

```
Disk /dev/hda: 9729 cylinders, 255 heads, 63 sectors/track
Units = cylinders of 8225280 bytes, blocks of 1024 bytes, counting
from 0

   Device Boot Start     End   #cyls    #blocks   Id  System
/dev/hda1   *      0+     12      13-    104391   83  Linux
/dev/hda2         13    9728    9716   78043770   8e  Linux LVM
/dev/hda3          0       -       0          0    0  Empty
/dev/hda4          0       -       0          0    0  Empty
```

For a Linux/Windows dual-boot disk, the output might be:

```
Disk /dev/sda: 19457 cylinders, 255 heads, 63 sectors/track
Units = cylinders of 8225280 bytes, blocks of 1024 bytes, counting
from 0

   Device Boot Start     End   #cyls    #blocks   Id  System
/dev/sda1          0+    891     892-   7164958+  27  Unknown
/dev/sda2   *    892   10196    9305   74742412+   6  FAT16
/dev/sda3      10197   10209      13     104422+  83  Linux
/dev/sda4      10210   19456    9247   74276527+   5  Extended
/dev/sda5      10210+  19456    9247-  74276496   8e  Linux LVM
```

Note that the FAT16 partition is actually formatted as NTFS. So one cannot trust the system labels to tell you what the actual filesystem is. However, most Windows partitions will list some recognizable label, such as FAT16, W95 FAT32, NTFS Volume Set, or the like.

Creating the Image

The basic command to use is:

```
bash# dd if=[device] of=[imagename]conv=noerror
```

dd reads in from one file (if, for input file), optionally runs a conversion filter, and then writes out to another. If the output file (of) is not specified, the data is written to standard output. The conv=noerror instructs the program to continue reading in case of an error, which may be necessary of the drive has been damaged.

Note also that the above command does a byte for byte copy of the hard drive, skipping any bytes that cannot be read. This can take a very long time for a hard drive of any size, especially if caching is disabled. It is not uncommon for the process to take several hours, and I have seen it take more than a day.

Most data recovery documents come with a standard warning: do not try to make your image in the same filesystem you are recovering from. While this may seem self-evident, evidently it happens frequently enough for the warning to be standard and so it is worth mentioning as a reminder.

Writing the Image to a File in a Mounted Filesystem

The simplest form of the command above might be:

```
bash# dd conv=noerror if=/dev/sda1 of=/root/recovery.img
```

This example copies the contents of /dev/sda1 to /root/recovery.img. The filesystem can then be mounted using a standard loopback device in Linux.

Writing to a Filesystem on Another System

Because dd can write to standard output, it can be used to write to a file on a remote system using ssh. In order to reduce the network overhead, we can optionally compress the contents and then uncompress them on the other side:

```
bash# dd conv=noerror if=/dev/sda1 | ssh user@host 'cat > recover.img'
```

To compress the stream in transit, we can enter the following (all on one line):

```
bash# dd conv=noerror if=/dev/sda1 | gzip |  ssh user@host 'gunzip >
recover.img'
```

In general, it is not a good idea to store the image compressed on a hard drive for recovery purposes because one cannot operate directly on it.

Mounting the Image

Once the image has been made, make two directories, one for the mountpoint and one for the destination:

```
bash# mkdir recover_src recover_dst
```

Then you can mount the filesystem with the following command:

```
bash# mount -t [type] -o loop /path/to/image recover_src
```

For example. assuming the command above was used to create the image in /root/recovery.img, and that it is NTFS, you can use:

```
bash# mount -t ntfs -o loop /root/recovery.img recover_src
```

All FAT filesystems would use the -t vfat flag.

A Note about NTFS Drivers and Linux NTFS Support

Not all Linux distributions come with NTFS drivers even though read-only drivers are a part of the current Linux source code. Although Knoppix does have such support, Fedora does not. Often you can locate drivers on third party web sites. In general, recompiling the Linux kernel is more trouble than you want to go through for something like this. If you can't find the drivers, use Knuppix.

NTFS filesystems can only be read on Linux without compromising safety-- writing to them is still considered dangerous and may result in data corruption or loss. Most of the drivers available only offer read-only support, and so getting read-write support is well beyond the scope of this paper.

Recovering the Data to a Safe Place

Once the loop image is mounted you can copy files using the cp or similar command into the recover_dst folder for later conversion into CDROM's, and the like.

Alternatively, one could use another USB drive formatted using FAT32, mount that on recover_dst, and continue to recover files in this way.

In general, I like to archive filesystem images (compressed usually) for a period of time so that I can ensure that everything that I needed to get off them has been saved before I disregard them.

Quick Notes on mkisofs and cdrecord

Often times, it may be useful to write the backed up files to a CDROM. Doing so requires the use of two important utilities: mkisofs and cdrecord.

CDROM's use a filesystem specified in the ISO 9660 standard. This filesystem has very few features and is, by itself incredibly limiting (no long file names, no permissions, case insensitivity in file names, etc). In order to make the filesystem more usable on modern operating systems, two sets of extensions were created: the Joliet extensions used by Windows, and the Rock Ridge extensions usually supported on UNIX.

In general, it is recommended that you use both sets of extensions in order to best integrate the CD across platforms.

The command to create an iso image named "recovery.iso" from the contents of the recovery_dst folder (with both Joliet and Rock Ridge extensions) would be:

```
bash# mkisofs -JR -o recovery.iso recovery_dst
```

Once one has created the image, it can be burned to a CD-RW drive using the cdrecord program. The simplest form of the cdrecord command might be:

```
bash# cdrecord dev=/dev/cdrom --data recovery.iso
```

If you want more progress information, you can add a -v argument.

When you are done, be sure to verify the media by mounting it and attempting to read the directory. I have generally found that some CD-R disks do not handle high burn rates well, and burning at the maximum rate may cause the entire disk to be bad. If this occurs, I have found that slowing the burn speed down to 4x usually solves the problem.

The similar command for cdrecord, asking for verbose output and a write speed of 4x is:

```
bash# cdrecord dev=/dev/cdrom speed=4 -v --data recovery.iso
```

Tools of Last Resort

In some cases, the filesystem may be too badly damaged to get the data out through standard means. Perhaps the disk was formatted, or the drive is failing. Either way, there are a few tools which can help resolve these cases given a fair bit more time, drive space, and patience.

The Coroner's Toolkit

The Coroner's Toolkit is a somewhat morbidly named set of tools for post-mortem analysis of a UNIX system (after a break-in). It is designed to discover data or programs which may not be visible to the operating system through the normal file interfaces. It is available at http://www.porcupine.org/forensics/tct.html.

The Coroner's Toolkit has a few utilities which can come in extremely handy in data recovery. Of these utilities, 'lazarus' is the most useful for cross-platform data recovery. This particular utility sorts through a data stream of unknown structure and attempts to reconstruct files from it. Sometimes it will only find fragments which will need to be reassembled later. Often file fragments from deleted files (where parts of the file have not yet been overwritten) may be recovered.

While lazarus and the rest of the TCT represent powerful tools for security analysis and data recovery, they also require a great deal of time, space, patience, and knowledge to use effectively. Their use is beyond the scope of this paper.

Data Recovery Specialists

When the drive appears to be failing or has been physically damages (for example. dropped on the floor), the best course of action if the data is of value is to hire a dedicated data recovery firm to handle this. Such firms rely not only on software methods similar to what I have described but generally also have the ability to disassemble the hardware in order to retrieve data from a physically damaged disk.

I have other customers who have had the drive heads come off the spindles and grind into the platters, who have had all important data recovered. Unfortunately this option is usually quite expensive and rates can vary significantly from region to region.

Final Thoughts

Data recovery is a huge field, and it is impossible to cover it in any significant depth in a short paper. However, I hope that this provides good information about how to use Linux to recover data from minor to moderate failures.

About the Author

Chris Travers owns Metatron Technology Consulting, a company helping customers run open source software on any operating system. He has used these techniques for the last eight years to recover data from floppy and hard drives.

Copyright