Securely wipe disk

Wiping a disk is done by writing new data over every single bit.

Note: References to "disks" in this article also apply to loopback devices.

Contents

Related articles

Securely wipe disk/Tips and tricks

File recovery

Benchmarking/Data storage devices

Frandom

Disk encryption#Preparing the disk

dm-crypt

Common use cases

Wipe all data left on the device

The most common usecase for completely and irrevocably wiping a device will be when the device is going to be given away or sold. There may be (unencrypted) data left on the device and you want to protect against simple forensic investigation that is mere child's play with for

example **File recovery** software.

If you want to quickly wipe everything from the disk, `/dev/zero` or simple patterns allow maximum performance while adequate randomness can be advantageous in some cases that should be covered up in **#Data remanence**.

Every overwritten bit means to provide a level of data erasure not allowing recovery with normal system functions (like standard ATA/SCSI commands) and hardware interfaces. Any file recovery software mentioned above then would need to be specialized on proprietary storage-hardware features.

In case of a HDD data recreation will not be possible without at least undocumented drive commands or fiddling about the device's controller or firmware to make them read out for example reallocated sectors (bad blocks that **S.M.A.R.T.** retired from use).

There are different wiping issues with different physical storage technologies, most notably all Flash memory based devices and older magnetic storage (old HDD's, floppy disks, tape).

Preparations for block device encryption

If you want to prepare your drive to securely set up **Disk encryption#Block device encryption** inside the wiped area afterwards you really should use **#Random data** generated by a trusted cryptographically strong random number generator (referred to as RNG in this article from now on).

See also **Wikipedia:Random number generation**.

Warning: If Block device encryption is mapped on a partition that contains anything else than random/encrypted data, disclosure of usage patterns on the encrypted drive is possible and weakens the encryption being comparable with filesystem-level-encryption. Never use `/dev/zero`, simple patterns (badblocks, eg.) or other unrandom data before setting up Block device encryption if you are serious about it!

Data remanence

See also **Wikipedia:Data remanence**.

The residual representation of data may remain even after attempts have been made to remove or erase the data.

Residual data may get wiped by writing (random) data to the disk with a single or even more than one iteration. However, more than one iteration may not significantly decrease the possibility to reconstruct the data of hard disk drives. See **#Residual magnetism**.

Operating system, programs and filesystem

The operating system, executed programs or **journaling file systems** may copy your unencrypted data throughout the block device. When writing to plain disks this should only be relevant in conjunction with one of the above.

If the data can get exactly located on the disk and was never copied anywhere else, wiping with random data can be thoroughgoing and impressively quick as long there is enough entropy in the pool.

A good example is cryptsetup using `/dev/urandom` for **wiping the LUKS keyslots**.

Hardware-specific issues

Flash memory

Write amplification and other characteristics make Flash memory (explicitly including SSDs) a stubborn target for reliable wiping. As there is a lot of transparent abstraction in between data as seen by a device's controller chip and the operating system sight data is never overwritten in place and wiping particular blocks or files is not reliable.

Other "features" like transparent compression (all SandForce SSD's) can compress your /dev/zero or pattern stream so if wiping is fast beyond belief this might be the case.

Disassembling Flash memory devices, unsoldering the chips and analyzing data content without the controller in between is feasible without difficulty using **simple hardware (http://www.flash-extractor.com/manual/reader_models/)**. Data recovery companies do it for cheap money.

For more information see:

- **SSD memory cell clearing**
- **Reliably Erasing Data From Flash-Based Solid State Drives (http://www.usenix.org/events/fast11/tech/full_papers/Wei.pdf)**.
- **#Select a target**

Marked Bad Sectors

If a hard drive marks a sector as bad, it cordons it off, and the section becomes impossible to write to via software. Thus a full overwrite would not reach it. However because of block sizes, these sections would only amount to a few theoretically recoverable KB.

Residual magnetism

A single, full overwrite with zeros or random data does not lead to any recoverable data on a modern high-density storage device.**[1] (http://www.howtogeek.com/115573/htg-explains-why-you-only-have-to-wipe-a-disk-once-to-erase-it/)** Indications otherwise refer to single residual bits; reconstruction of byte patterns is generally not feasible.**[2] (https://web.archive.**

org/web/20120102004746/http://www.h-online.com/newsticker/news/item/Secure-deletion-a-single-overwrite-will-do-it-739699.html) See also [3] (https://www.google.com/search?tbs=bks:1&q=isbn:9783540898610), [4] (http://security.stackexchange.com/questions/26132/is-data-remanence-a-myth/26134#26134) and [5] (http://www.nber.org/sys-admin/overwritten-data-guttman.html).

Select a target

Note: Fdisk will not work on GPT formatted devices. Use gdisk (`gptfdisk (https://www.archlinux.org/packages/?name=gptfdisk)`) instead.

Use fdisk to locate all read/write devices the user has read acess to.

Check the output for lines that start with devices such as `/dev/sd"X"`.

This is an example for a HDD formatted to boot a linux system:

```
# fdisk -l
```

```
Disk /dev/sda: 250.1 GB, 250059350016 bytes, 488397168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00ff784a

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048      206847      102400   83  Linux
/dev/sda2          206848   488397167   244095160   83  Linux
```

Or the Arch Install Medium written to a 4GB USB thumb drive:

```
# fdisk -l

Disk /dev/sdb: 4075 MB, 4075290624 bytes, 7959552 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x526e236e

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *           0      802815      401408   17  Hidden HPFS/NTFS
```

If you are worried about unintentional damage of important data on the primary computer, consider using an isolated environment such as a virtual environment (VirtualBox, VMWare, QEMU, etc...) with direct connected disk drives to it or a single computer only with a storage disk(s) that need to be wiped booted from a **Live Media**(USB, CD, PXE, etc...) or use a script to **prevent wiping mounted partitions by typo**.

Select a block size

See also **Wikipedia:Dd (Unix)#Block size**, **blocksize io-limits (http://people.redhat.com/msnitzer/docs/io-limits.txt)**.

If you have an **Advanced Format** hard drive it is recommended that you specify a block size larger than the default 512 bytes. To speed up the overwriting process choose a block size matching your drive's physical geometry by appending the block size option to the *dd* command (i.e. `bs=4096` for 4KB).

fdisk prints physical and logical sector size for every disk. Alternatively sysfs does expose information:

```
/sys/block/sdX/size
/sys/block/sdX/queue/physical_block_size
/sys/block/sdX/queue/logical_block_size
/sys/block/sdX/sdXY/alignment_offset
/sys/block/sdX/sdXY/start
/sys/block/sdX/sdXY/size
```

Warning: These methods show the block size the drive reports to the kernel. However, many Advanced Format drives incorrectly understate the physical block size as 512.

Tip: This script helps to calculate parameters to wipe a device/partition with dd **genwipe.sh (https://aur.archlinux.org/packages/genwipe.sh/)**[AUR], e.g. `genwipe.sh /dev/sd"XY"` .

Calculate blocks to wipe manually

In the following the determination of the data area to wipe is done in an example.

A block storage devices contains sectors and a size of a single sector that can be used to calculate the whole size of device in bytes. You can do it by multiplying sectors with size of the sector.

As an example we use the parameters with the *dd* command to wipe a partition:

```
# dd if=data_source of=/dev/sd"X" bs=sector_size count=sector_number seek=partitions_start_sector
```

Here you will see only a part of output of `fdisk -l /dev/sdX` with root, showing the example partition information:

```
Device     Boot      Start        End         Sectors     Size  Id Type
/dev/sd"XA"            2048         3839711231  3839709184  1,8T  83 Linux
/dev/sd"XB"            3839711232   3907029167  67317936    32,1G  5 Extended
```

The first line of the *fdisk* output shows the disk size in bytes and logical sectors:

```
Disk /dev/sd"X": 1,8 TiB, 2000398934016 bytes, 3907029168 sectors
```

To calculate size of a single logical sector use `echo $((2000398934016 / 3907029168))` or use data from the second line of *fdisk* output:

```
Units: sectors of 1 * 512 = 512 bytes
```

To calculate physical sectors that will make it work faster we can use the third line:

```
Sector size (logical/physical): 512 bytes / 4096 bytes
```

To get disk size in the physical sectors you will need the known disk size in bytes divided with size of a single physical sector `echo $((2000398934016 / 4096))` , you can get size of the storage device or partition on it even with the `blockdev --getsize64 /dev/sd"XY"`

command.

Note:

- In the examples below we will use the logical sector size.
- You can even wipe unallocated disk space with a `dd` command by calculating the difference between the end of one and start of the next partition.

To wipe partition `/dev/sd"XA"` the example parameters with logical sectors would be used like this:

```
Start=2048
End=3839711231
BytesInSector=512
```

By using the starting address of the partition on the device by defining it in the `seek=` option

```
# dd if=data_source of=/dev/sd"X" bs=${BytesInSector} count=${End} seek=${Start}
```

By using the partitions name

```
LogicalSectors=3839709184
```

```
# dd if=data_source of=/dev/sd"XA" bs=${BytesInSector} count=${LogicalSectors}
```

Or, to wipe the whole disk by using physical sectors:

```
AllDiskPhysicalSectors=488378646
PhysicalSectorSizeBytes=4096
```

```
# dd if=data_source of=/dev/sd"X" bs=${PhysicalSectorSizeBytes} count=${AllDiskPhysicalSectors} seek=0
```

Note: The `count=` option not necessary when wiping the physical limited area e.g. `sd"XY"` or `sd"X"` from begin to the end but will show an error about out of free space when will try to write outside of limits.

Select a data source

As just said If you want to wipe sensitive data you can use anything matching your needs.

If you want to setup block device encryption afterwards, you should always wipe at least with an encryption cipher as source or even pseudorandom data.

For data that is not truly random your disk's writing speed should be the only limiting factor. If you need random data, the required system performance to generate it may extremely depend on what you choose as source of entropy.

Non-random data

Overwriting with `/dev/zero` or simple patterns is considered secure in most resources. In the case of current HDD's it should be sufficient for fast disk wipes.

Warning: A drive that is abnormally fast in writing patterns or zeroing could be doing transparent compression. It is obviously presumable not all blocks get wiped this way. Some **#Flash memory** devices do "feature" that.

Pattern write test

#Badblocks can write simple patterns to every block of a device and then read and check them searching for damaged areas (just like memtest86* does with memory).

As the pattern is written to every accesible block this effectively wipes the device.

Random data

For differences between random and pseudorandom data as source, please see **Random number generation**.

Note: Data that is hard to compress (random data) will get written slower, if the drive logic mentioned in the **#Non-random data** warning tries compressing it. This should not lead to **#Data remanence** though. As maximum write-speed is not the performance-bottleneck it can get completely neglected while wiping disks with random data.

Encrypted data

When preparing a drive for full-disk encryption, sourcing high quality entropy is usually not necessary. The alternative is to use an encrypted datastream. For example, if you will use AES for your encrypted partition, you would wipe it with an equivalent encryption cipher prior to creating the filesystem to make the empty space not distinguishable from the used space.

Overwrite the target

The chosen drive can be overwritten with several utilities, make your choice. If you only want to wipe a single file, **Securely wipe disk/Tips and tricks#Wipe a single file** has considerations in addition to the utilities mentioned below.

By redirecting output

The redirected output can be used both for creation of the files to rewrite free space on the partition, wipe the whole device or a single partition on it.

In the following are examples that can be used to rewrite the partition or a block device by redirecting **stdout (http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-3.html)** from other utilities:

```
$ cat /dev/urandom > /dev/sd"XY"
```

```
cat: write error: No space left on device
```

```
$ xz -z0 /dev/urandom -c > /dev/sd"XY"
```

```
xz: (stdout): Write error: No space left on device
```

```
$ dd if=/dev/urandom > /dev/sd"XY"
```

```
dd: writing to ‘standard output’: No space left on device
20481+0 records in
20480+0 records out
10485760 bytes (10 MB) copied, 2.29914 s, 4.6 MB/s
```

The file copy command `cp` can also be used to rewrite the device, because it ignores the type of the destination:

```
$ cp /dev/urandom /dev/sd"XY"
```

```
cp: error writing ‘/dev/sd"XY"’: No space left on device
cp: failed to extend ‘/dev/sd"XY"’: No space left on device
```

To show speed and time you can use **pv (https://www.archlinux.org/packages/?name=pv)**:

```
# pv --timer --rate --stop-at-size -s "$(blockdev --getsize64 /dev/sd"XY" )" /dev/zero > /dev/sd"XY"
```

dd

See also **Core utilities#dd**.

Warning: There is no confirmation regarding the sanity of this command so **repeatedly check** that the correct drive or partition has been targeted. Make certain that the `of=...` option points to the target drive and not to a system disk.

Zero-fill the disk by writing a zero byte to every addressable location on the disk using the **/dev/zero** stream.

```
# dd if=/dev/zero of=/dev/sdX bs=4096
```

Or the **/dev/urandom** stream:

```
# dd if=/dev/urandom of=/dev/sdX bs=4096
```

The process is finished when dd reports `No space left on device` and returns control back:

```
dd: writing to ‘/dev/sdb’: No space left on device
7959553+0 records in
7959552+0 records out
4075290624 bytes (4.1 GB) copied, 1247.7 s, 3.3 MB/s
```

To speed up wiping a large drive, see also:

- **Securely wipe disk/Tips and tricks#dd - advanced example** which uses OpenSSL,

- **Securely wipe disk/Tips and tricks#Using a template file** which wipes with non-random preset data(e.g. overwrite a whole disk with a single file) but is very fast
- **Dm-crypt/Drive preparation#dm-crypt specific methods** which uses dm-crypt.

wipe

Specialized on wiping files and is available as the `wipe (https://www.archlinux.org/packages/?name=wipe)` package. To make a quick wipe of a destination you can use something like:

```
$ wipe -r -q /path/to/wipe
```

See also `wipe(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/wipe.1)`.

shred

shred **(https://www.gnu.org/software/coreutils/manual/html_node/shred-invocation.html)** (from the `coreutils (https://www.archlinux.org/packages/?name=coreutils)` package) is a Unix command that can be used to securely delete individual files or full devices so that they can be recovered only with great difficulty with specialised hardware, if at all. By default *shred* uses three passes, writing **pseudo-random data** to the device during each pass. This can be reduced or increased.

The following command invokes shred with its default settings and displays the progress.

```
# shred -v /dev/sdX
```

Alternatively, shred can be instructed to do only one pass, with entropy from e.g. `/dev/urandom`.

```
# shred --verbose --random-source=/dev/urandom -n1 /dev/sdX
```

Badblocks

For letting badblocks (from the **e2fsprogs (https://www.archlinux.org/packages/?name=e2fsprogs)** package) perform a disk wipe, a destructive **read-write test** has to be done:

```
# badblocks -c <NUMBER_BLOCKS> -wsv /dev/<drive>
```

hdparm

Warning: Do not attempt to issue a Secure Erase ATA command on a device connected through USB; see **https://ata.wiki.kernel.org/index.php/ATA_Secure_Erase** and **http://www.tomshardware.co.uk/answers/id-1984547/secure-erase-external-usb-hard-drive.html** for details.

hdparm supports **ATA Secure Erase (http://tinyapps.org/docs/wipe_drives_hdparm.html)**, which is functionally equivalent to zero-filling a disk. It is however handled by the hard-drive firmware itself, and includes "hidden data areas". As such, it can be seen as a modern-day "low-level format" command. **SSD** drives reportedly achieve factory performance after issuing this command, but may not be sufficiently wiped (see **#Flash memory**).

Some drives support **Enhanced Secure Erase**, which uses distinct patterns defined by the manufacturer. If the output of `hdparm -I` for the device indicates a manifold time advantage for the **Enhanced** erasure, the device probably has a hardware encryption feature and the wipe will be performed to the encryption keys only.

For detailed instructions on using ATA Secure Erase, see the **Linux ATA wiki (https://ata.wiki.kernel.org/index.php/ATA_Secure_Erase)**.

See also

- **Wipe free space in Linux (http://superuser.com/questions/19326/how-to-wipe-free-disk-space-in-linux)**