Securely wipe disk

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

Related articles

- Securely wipe disk/Tips and tricks
- File recovery
- Benchmarking/Data storage devices
- Data-at-rest encryption#Preparing the disk
- dm-crypt

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

1 Common use cases

1.1 Wipe all data left on the device

The most common usecase for completely and irrevocably wiping a device is 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 tinkering with 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 HDDs, floppy disks, tape).

1.2 Preparations for block device encryption

To prepare a drive for **block device encryption** inside the wiped area afterwards, it is recommended to use **#Random data** generated by a 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 non-random or unencrypted data, the encryption is weakened and becomes comparable to filesystem-level encryption: disclosure of usage patterns on the encrypted drive becomes possible. Therefore, do not fill space with zeros, simple patterns (like badblocks) or other non-random data before setting up block device encryption if you are serious about it.

2 Data remanence

See also **Wikipedia:Data remanence**. The representation of data may remain even after attempts have been made to remove or erase the data.

2.1 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 be 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**.

2.2 Hardware-specific issues

2.2.1 Flash memory

Write amplification and other characteristics make Flash memory, 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 SSDs) can compress your zeros or repetitive patterns, so if wiping is fast beyond belief this might be the cause.

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:

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

2.2.2 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 KiB.

2.2.3 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. Note that repeating the operation should not be necessary nowadays. **[1] (https://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-wil** l-do-it-739699.html) See also [3] (https://www.google.com/search?tbs=bks:1&q=isbn:9783540898610), [4] (https://security.stackexchange.com/questions/26132/is-data-remanence-a-myth/26134#26134) and [5] (https://www.nber.org/sys-admin/overwritten-data-guttman.html).

3 Select a target

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

Check the output for lines that start with devices such as `/dev/sdX`.

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 another example with the Arch Linux image 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**.

4 Select a data source

To wipe sensitive data, one can use any data pattern matching the needs.

4.1 Zeros

Overwriting with `/dev/zero` or simple patterns is considered secure in most situations. With today's HDDs, it is deemed appropriate and fast for disk wiping.

However, 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.

To setup block device encryption afterwards, one should wipe the area with random data (see next section) to avoid weakening the encryption.

Warning: Subject to compression and to be used carefully with flash memory and SSDs, to be avoided for block encryption preparation as stated above.

4.2 Random data

`/dev/urandom` can be used as a fast and secure source of cryptographically secure pseudorandom data from the Linux kernel. For more details about sources of random and pseudorandom data, see **Random number generation**.

In the past when the kernel's random number generator was slow, a common alternative for pseudorandom data generation was to use an encrypted datastream, such as by encrypting `/dev/zero` with a random key. While this should in theory be secure, it no longer presents any advantages over the kernel's new, faster random number generator, and there is a risk that the temporary key may accidentally be saved someplace.

5 Select a block size

See also **Wikipedia:Dd (Unix)#Block size**, **blocksize io-limits (https://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 4 KiB).

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: The script **`genwipe.sh (https://aur.archlinux.org/packages/genwipe.sh/)`** AUR helps to calculate parameters to wipe a device/partition with *dd*, e.g. `genwipe.sh /dev/sdXY`.

5.1 Calculate blocks to wipe manually

Block storage devices are divided in sectors, and the size of a single sector can be used to calculate the size of the entire device in bytes. To do so, multiply the number of sectors by the drive sector size.

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

```
# dd if=data_source of=/dev/sdX bs=sector_size count=sector_number seek=partitions_start_sector status=prog
ress
```

Here, to illustrate with a practical example, we will show the output of the *fdisk* command on the partition `/dev/sdX`:

```
# fdisk -l /dev/sdX
```

```
Disk /dev/sdX: 1.8 TiB, 2000398934016 bytes, 3907029168 sectors
Disk model: ST3500413AS
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
...
Device     Boot      Start        End          Sectors     Size  Id Type
/dev/sdX1            2048         3839711231   3839709184  1,8T  83 Linux
/dev/sdX2            3839711232   3907029167   67317936    32,1G  5 Extended
```

- The first line of the *fdisk* output shows the disk size in bytes and in logical sectors.
- The size in bytes of the storage device or of the partition can also be obtained with the command `blockdev --getsize64 /dev/sdXY`.
- The *Units* line of the *fdisk* output shows the size of single logical sector; the logical sector size can also be derived from the number of bytes divided by the number of logical sectors, here use: `echo $((2000398934016 / 3907029168))`.
- To know the physical sector size in bytes (that will make it work faster), we can use the next line.
- To get the disk size in physical sectors, one can divide the disk size in bytes by the size of a single physical sector, here `echo $((2000398934016 / 4096))`,

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/sdX1`, the example parameters with logical sectors would be used like follows.

- By using the starting address of the partition on the device using the `seek=` parameter:

```
# dd if=data_source of=/dev/sdX bs=${BytesInSector} count=${End - Start} seek=${Start} status=progress
```

with `Start=2048`, `End=3839711231` and `BytesInSector=512`.

- Or by using the partitions size in logical sectors:

```
# dd if=data_source of=/dev/sdX1 bs=${BytesInSector} count=${LogicalSectors} status=progress
```

with `LogicalSectors=3839709184`.

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

```
# dd if=data_source of=/dev/sdX bs=${PhysicalSectorSizeBytes} count=${AllDiskPhysicalSectors} seek=0 status
=progress
```

with `AllDiskPhysicalSectors=488378646` and `PhysicalSectorSizeBytes=4096`.

Note: The `count=` option is not necessary when wiping all the physical area e.g. `sdXY` or `sdX` from the start to the end but it shows an error about out of free space when it tries to write outside the limits.

6 Overwrite the target

You can choose from several utilities to overwrite a drive. 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.

6.1 By redirecting output

The redirected output can be used to create files, rewrite free space on the partition, and to wipe the whole device or a single partition on it. The examples here use `/dev/zero` to zero the device, but `/dev/urandom` may be substituted if a random wipe is desired.

The following examples show how to rewrite the partition or a block device by redirecting **stdout (https://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-3.html)** from other utilities:

```
# cat /dev/zero > /dev/sdXY
```

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

```
# xz -z0 /dev/zero -c > /dev/sdXY
```

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

```
# dd if=/dev/zero > /dev/sdXY
```

```
dd: writing to ‘standard output’: No space left on device
20481+0 records in
20480+0 records out
10485760 bytes (10 MB, 10 MiB) 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/zero /dev/sdXY
```

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

To display progress status and metrics you can use **`pv (https://archlinux.org/packages/?name=pv)`**:

```
# pv --progress --timer --eta --rate --bytes --stop-at-size -s "$(blockdev --getsize64 /dev/sdXY )" /dev/ze
ro > /dev/sd"XY"
```

6.2 dd

See also **dd** and **Securely wipe disk/Tips and tricks#Wipe a single file**.

Warning: There is no confirmation regarding the sanity of this command so **double-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 status=progress
```

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

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

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

```
dd: writing to ‘/dev/sdX’: No space left on device
7959553+0 records in
7959552+0 records out
4075290624 bytes (4.1 GB, 3.8 GiB) 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.

6.3 wipe

A program specialized on wiping files. It is available as part of the **`wipe (https://archlinux.org/packages/?name=wipe)`** package. To make a quick wipe of a destination, you can use something like:

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

See also **`wipe(1) (https://man.archlinux.org/man/wipe.1)`**. The tool was last updated in 2009. Its **SourceForge page (http://wipe.sourceforge.net/)** suggests that it is **currently unmaintained**.

6.4 shred

***shred* (https://www.gnu.org/software/coreutils/manual/html_node/shred-invocation.html)** (from the **`coreutils (https://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
```

Shred can also be used on a single partition, e.g. to wipe the first partition use `shred -v /dev/sdX1`.

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

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

6.5 scrub

scrub (https://github.com/chaos/scrub) iteratively writes patterns on files or disk devices to make retrieving the data more difficult.

The following command invokes scrub with the default settings, in mode 1, overwriting the target device using patterns compliant with NNSA Policy Letter NAP-14.x. This is the most effective method.

```
$ scrub /dev/sdX
```

The following command invokes scrub with the default settings, in mode 2, overwriting the target file using patterns compliant with NNSA Policy Letter NAP-14.x, rounding the bytes written up to fill out the last file system block. Note that there are caveats for this mode, see the *manual* (https://linux.die.net/man/1/scrub) for further details.

```
$ scrub /path/to/file # where file is a regular file
```

The following command invokes scrub with the default settings, in mode 3, creating a directory and filling it with files until the file system is full. The files are then scrubbed using patterns compliant with NNSA Policy Letter NAP-14.x, rounding the bytes written up to fill out the last file system block. Note that there are caveats for this mode, see the *manual* (https://linux.die.net/man/1/scrub) for further details.

```
$ scrub /path/to/dir # where dir is a new directory name.
```

For further usage and information, see the *manual* (https://linux.die.net/man/1/scrub).

6.6 badblocks

The tool badblocks from `e2fsprogs (https://archlinux.org/packages/?name=e2fsprogs)` is able to perform destructive read-write test, effectively wiping the device. By default, it performs four passes and can take a long time.

```
# badblocks -wsv /dev/device
```

6.7 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 this answer (https://web.archive.org/web/20180711170859/http://www.tomshardware.co.uk/answers/id-1984547/secure-erase-external-usb-hard-drive.html) for details.

hdparm supports ATA Secure Erase (https://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 many-fold 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 Solid state drive/Memory cell clearing and the Linux ATA wiki (https://ata.wiki.kernel.org/index.php/ATA_Secure_Erase).

6.8 blkdiscard

See Solid state drive/Memory cell clearing#Common method with blkdiscard

7 See also

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

-