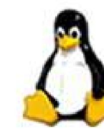
pyFLAG



LCA 2005
Security
Miniconf

RAID Reconstruction

And the search for the Aardvark

Dr Michael Cohen

This talk does not represent my Employer

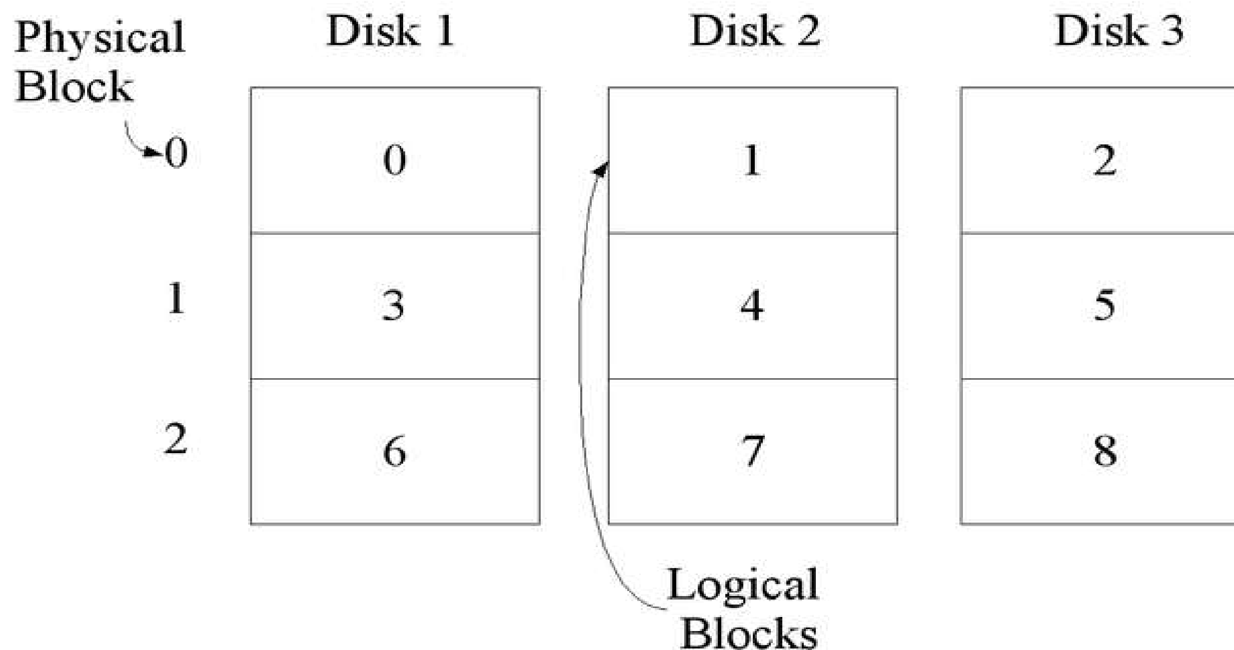
April 2005

pyFLAG

LCA 2005 Security Miniconf

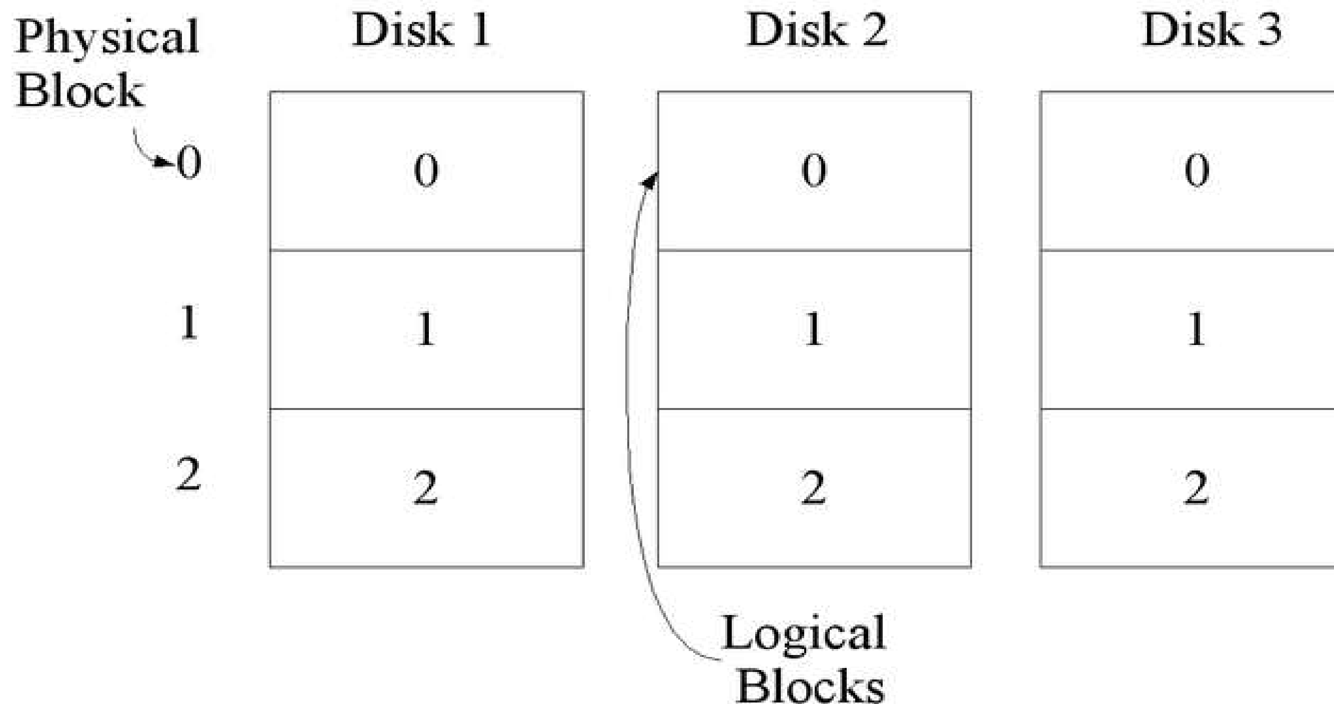
What is RAID?

- **RAID 0: Striping**
  - Improves performance due to parallel disk access
  - No redundancy – Lose a single disk=lose data
  - Capacity of array = n * capacity of each disk



RAID 1: Mirroring

- Improves read speed – not write speed
- Full redundancy – May lose any number of disks but one
- Capacity of array = capacity of one disk



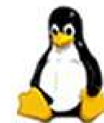
RAID 5: Striping with parity

- Strikes a balance between redundancy, speed and capacity
- Data is stripped across all disks and includes a parity block:
  - A xor B xor C = 0
- Can lose one disk from the array and recover the data.

RAID-5 Configuration

Physical Block	Disk 1	Disk 2	Disk 3
0	0	1	P(1+2)
1	2	P(2+3)	3
2	P(4+5)	4	5

Logical Blocks



Why would we care?

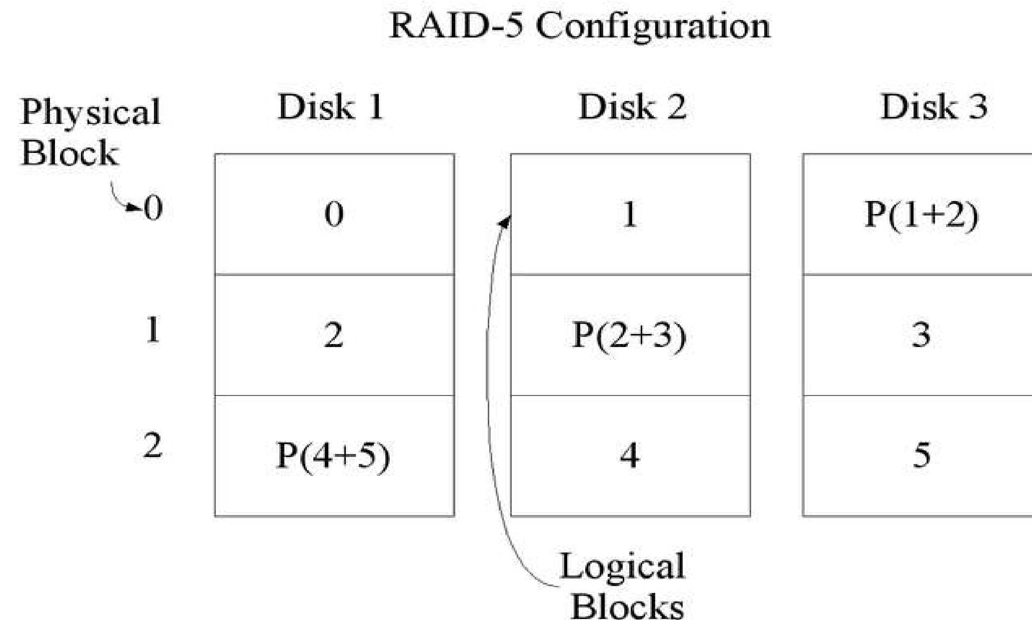
- **Image acquisition:**
  - RAID arrays are common in server class machines
  - Traditional techniques involve booting the original hardware and imaging over network/USB
    - This is typically slow
    - Cant use imaging hardware like Logicube etc.
- **Data recovery:**
  - Some controllers can mark disks as bad when they detect a fault with the disk:
    - Often the fault is intermittent.
    - A drive may be repairable.

Sounds easy?

- **We can just reassemble the striping pattern as in the previous figure?**
  - Unfortunately there is no standard striping pattern – Each controller has its own pattern
  - Often the order of the disks in the array is nothing like the labeling on the disk
  - Sometimes we are missing one drive – we should be able to rebuild the array without it.

Definitions:

- **Block**
  - Logical Block
  - Physical Block
- **Array period**
  - Physical array period
  - Logical array period
  - Slot
- **Striping Map**



pyFLAG

LCA 2005
Security
Miniconf

How do we reassemble the array?

- **We need to establish the block size**
  - Look through the data for obviously disjoint dis-continuities

```
                Disk1               Disk2                Disk3
0x1dfe0 ..S(....Ns....wh s.S(.hre(....... s....hrefs....re
0x1dff0 ite(....(....s.. ...Kes..(.pos.|p places....posixp
0x1e000 ..self(....(.... .as.possible.so. .a.E..[siblM.so.
0x1e010 s5.../home/mic/p it.may.be.instal .A.maVH..EF...N.
0x1e020 yflag_website/py led.with.as.few. ...A.6..B..T.J.Y
                         .....
0x1efe0 pts.to.include.a ill.then.be.inde ......E.N..U..D.
0x1eff0 s.many.libraries xed.during.scann .E.A..R...R.....
0x1f000 ...Gedz....HI... ing...Usage:.loa tation/tutorials
0x1f010 K6....G....[.$.T d_dictionary.py. /index.html">Tut
0x1f020 46....SA.Ls.I..[ [Dictionary.file orials</a>.</td>
```

Demo

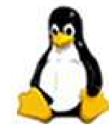
Establish the RAID map

- **Common pitfalls to avoid:**
  - Block that contain zeros will copy the parity block – so it will be difficult to tell them apart.
  - Regions of the disk containing random data can be indistinguishable from parity and are best avoided.
- **Bonus:**
  - Finding a dictionary makes life really easy because we are able to follow the block order alphabetically:
    - Search for Aardvark
  - For example /usr/share/dict/american-english is 900kb, with a 4k block size this will cover over contiguous blocks.

```
Block 30:
     0x1e000 ..self(....(.... .as.possible.so. .a.E..[siblM.so.
     0x1e010 s5.../home/mic/p it.may.be.instal .A.maVH..EF...N.

     0x1efe0 pts.to.include.a ill.then.be.inde ......E.N..U..D.
     0x1eff0 s.many.libraries xed.during.scann .E.A..R...R.....
Block 31:
     0x1f000 ...Gedz....HI... ing...Usage:.loa tation/tutorials
     0x1f010 K6....G....[.$.T d_dictionary.py. /index.html">Tut

     0x1ffe0 .L..Q.U;=-)gmBy$ olor="white">.<a k.rel="STYLESHEE
     0x1fff0 tJR..XD..j@..... .href="./Documen T".href="./defau
Block 32:
     0x20000 g.has.not.been.t .TF..SLO.Y..XLT. lt.css".type="te
     0x20010 ested.on.windows ..[..SMP*KF....M xt/css">.</head>

     0x20fe0 .wrong?</li>.</o N......L@.._hUC. nhance.portabili
     0x20ff0 l>.<div.class="a .G$.K.H*_._#.{N. ty.</p>.<p>PyFla
Block 33:
     0x21000 dmonition-answer <a.href="%(rootd X.O....RL.I.....
     0x21010 .admonition">.<p ir)sDocumentatio I.M.+......V_~U.
```

Build the RAID Map

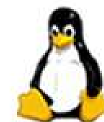
Block	Disk 1	Disk 2	Disk 3
30:	0	1	P
31:	P	2	3
32:	5	P	4
33:	6	7	P
34:	P	8	9



Determine the period

- **The period is the number of blocks before the pattern starts repeating**
  - Its usually easy to determine the period by looking at the parity.
  - In the previous case the period was 3 so the slot map was:

Block	Disk 1	Disk 2	Disk 3
30:	0	1	P
31:	P	2	3
32:	5	P	4



Recovering the partition table

- **Determine the start of the partition in the logical image:**
  - Usually there is a partition table at the start of the logical image:

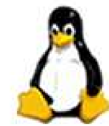
```
~/pyflag$ ./bin/mmls -t dos d1.dd
DOS Partition Table
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000001   0000000062   0000000062   Unallocated
02:  00:00   0000000063   0000009829   0000009767   Linux (0x83)
03:  00:01   0000009830   0000020479   0000010650   Win95 FAT32 (0x0C)
```

pyFLAG

LCA 2005
Security
Miniconf

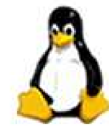
PyFlag Hooking infrastructure

- **PyFlag is a GPL forensic software**
  - There are many types of forensic images, in lots of different formats:
    - dd format (disk/partition)
    - Encase format
    - RAID Format
  - There are many different types of forensic tools
    - Sleuthkit
    - PyFlag
    - Lookback mounting through the kernel
    - md5sum, file, hexedit
    - Exgrep, foreman, strings



Putting them together

- **We need to use Forensic Tools on Forensic Images**
  - It would be best if we can use standard forensic tools on all kinds of images
  - We do not want to change the tools:
    - Although many tools are Open Source, it would be a nightmare to maintain patches against all of them.
  - We wish to enhance the functionality of arbitrary tools without touching their source code? Is this possible?



Hooking IO For fun and profit

- **We can do this by using a technique called library hooking:**
  - Consider a program:

```
main() {
   fd=open("somefile",O_RDONLY);
   read(fd,buffer,SIZE);
   close(fd);
}
```

How does this work?

Normal Operation

open()   open syscall

Program → C Library → Kernel

Hooked program

Program → Hooker → C Library → Kernel

open()   Driver
           calls open()

How does this work?

- The hooker library intercepts standard library calls related to IO (open, read, write etc)
- When the application wants to issue an IO function, the hooker uses an appropriate driver to access the raw image depending on its format
- The Hooker returns the data to the program as if the image was a simple linear block of data.
- Parameters to the hooker driver are passed in through environment variables.
- An iowrapper is responsible for arranging the hooker and its environment variables.
- The hooker can use a number of drivers (We call them subsystems).

pyFLAG

LCA 2005
Security
Miniconf

Example:

- **Let us see the hexdump of an Encase image:**

```
~/pyflag$ hexdump -C test.e01 | head
00000000  45 56 46 09 0d 0a ff 00  01 01 00 00 00 68 65 61  |EVF...ÿ......hea|
00000010  64 65 72 00 00 00 00 00  00 00 00 00 00 b2 00 00  |der..........².. |
00000020  00 00 00 00 00 a5 00 00  00 00 00 00 00 80 00 10  |.....¥..........|

~/pyflag$ ./bin/iowrapper -i ewf -filename=test.e01
-- hexdump -C foo | head
00000000  eb 3c 90 4d 53 44 4f 53  35 2e 30 00 02 01 01 00  |ë<.MSDOS5.0.....|
00000010  02 e0 00 40 0b f0 09 00  12 00 02 00 00 00 00 00  |.à.@.ð..........|
00000020  00 00 00 00 00 00 29 fc  02 29 08 4e 4f 20 4e 41  |......)ü.).NO NA|
00000030  4d 45 20 20 20 20 46 41  54 31 32 20 20 20 33 c9  |ME    FAT12   3É|
```

pyFLAG

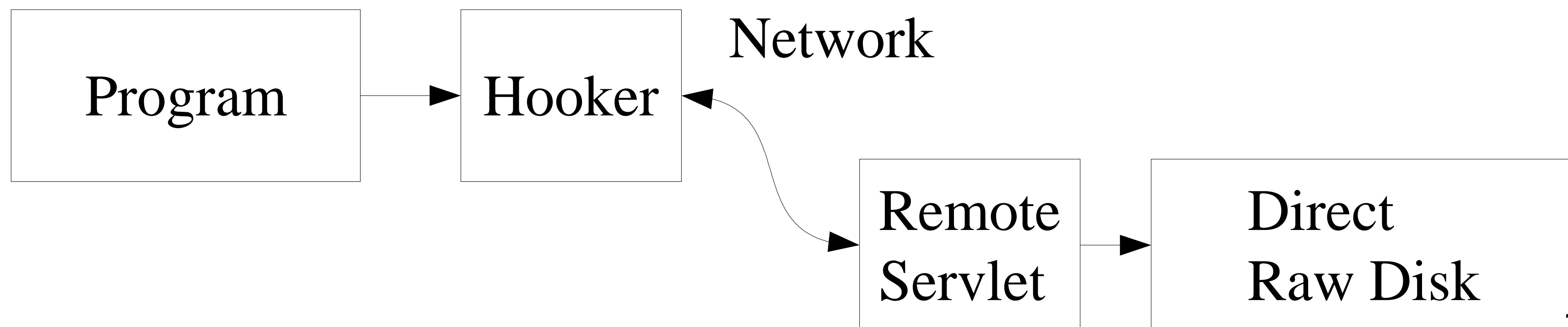
Example

- **Let us convert an Encase image to a dd image:**
  - The -f ensure we only hook files called foo – this allows dd to create the output file.
  - blocksize is 64k to ensure faster performance.

```
~/pyflag$ ./bin/iowrapper -i ewf
-filename=test.e01 -f foo -- dd if=foo
of=out.dd bs=64k
```

Remote Subsystem

- **We can trick local programs to read disk images directly off remote systems:**
  - Program being hooked receives data across the network from a remote servlet.
  - The servlet provides direct access to the raw disk on the remote machine



Remote Subsystem

- **Advantages:**
  - We do not need to copy the entire image prior to analysis – we can analyse the remote image un-intrusively in place.
  - Since the local program reads the filesystem structures directly, rather than relying on remote system's kernel system calls – we are immuned from kernel level rootkits on the remote system hiding files.
- **Disadvantages:**
  - Slow operation due to network latency.
  - Possibilities of race conditions since we are operating on a live system.

Example

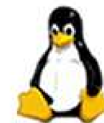
```
~/pyflag$./bin/iowrapper -i remote -host \
127.0.0.1 -user root -server_path  \
/bin/remote_server -device /dev/hdc -- mmls foo
```

```
DOS Partition Table
Sector: 0
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000001   0000000062   0000000062   Unallocated
02:  00:00   0000000063   0000096389   0000096327   Dell Utilities FAT (0xde)
03:  00:01   0000096390   0019647494   0019551105   NTFS (0x07)
```

Back to reassembling RAID:

- **Through the use of hooking, we can use arbitrary programs with the RAID driver transparently:**
  - The IO Hooker hooks system calls to enable other tools to work with the *reassembled* image.
  - Example:

./bin/iowrapper -i raid -blocksize 4k -slots 3 -offset 63s -map 0.1.P.P.2.3.5.P.4 -filenames d?.dd -- ./bin/fls -r -f linux-ext2 foo



LCA 2005
Security
Miniconf

Recover the Logical Image:

- **Use PyFlag to hook dd to copy the logical image to a dd image:**

```
./bin/iowrapper -i raid -blocksize 4k -slots 3 -offset 63s -map 0.1.P.P.2.3.5.P.4 -filenames d?.dd -- dd if=foo > /tmp/recovered.dd
```

Is there a shortcut?

- **Shortcuts make life easier**
  - Sometimes the shortcuts don't work
  - If you can save 2 hours by trying something that takes a few seconds it might be worth it.
- **PyFlag's hooker reassembles the array on the fly:**
  - The logical image is reassembles on an as needed basis by the external program.
  - It takes no time to try.
- **We can brute force the RAID parameters!!!!**

pyFLAG

Brute Forcing RAID Parameters

- **Can we tell when the parameters are wrong?**
  - We need a program that gives an indication of success.
  - We use it to guess the map from a library of possible maps
  - We need to assume a number of slots and a block size:

```
~/pyflag$ ./launch.sh utilities/raid_guess.py -o 63s -s 3 -b 4k d1.dd d2.dd d3.dd
```

What if one of the disks is dead?

- **PyFlag can recover a single disk from a RAID 5 array:**
  - If PyFlag fails to open the file, it assumes its missing:

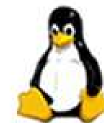
```
./bin/iowrapper -i raid -blocksize 4k -slots 3 -offset
63s -map 0.1.P.P.2.3.5.P.4 -filenames d1.dd d2.dd foo
-- ./bin/fls -r -f linux-ext2 foo
```

Set number of slots to 3
Set file number 0 as d1.dd
Set file number 1 as d2.dd
Set file number 2 as foo
Could not open file foo, marking as missing

What else can go wrong?

- **Sometimes there is a header at the start of the disk:**
  - Common with software RAID (e.g. Microsoft LDM, HP Smart Array)
  - You can find those by searching for the partition table:

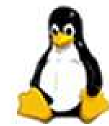
```
bash$ mmls -t dos header.dd
DOS Partition Table
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000001   0000000062   0000000062   Unallocated
02:  00:02   0000000063   0000016064   0000016002   Hibernation (0x12)1
```



How to find the header

- **We can guess the header size by trying to find the logical image's partition table:**

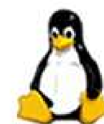
```
bash$ for i in `seq 0 4 1000`; do \
   echo trying offset ${i}k;  \
   ./bin/iowrapper -i advanced -offset ${i}k -filename  d1.dd -- \
   ./bin/mmls -t dos foo; \
done 2>&1 | less
```



Can we Hook the kernel?

- **Sometimes it would be nice to mount the image over the loopback device:**
  - This uses the kernel's own filesystem drivers – supports many more types of filesystems than Sleuthkit.
  - The kernel loopback driver does not use the C library to access the image – hence we need to hook in kernel space.
  - Fuse (Filesystems in user space) allows us to hook system calls in the kernel directly, passing them to a user space program to implement a virtual filesystem.

Filesystems in User Space



PyFlag's Fuse hooker

- **The Fuse hooker hooks IO in the kernel:**
  - Works on <u>all</u> programs due to very low level hooking.
  - Creates a pseudo file representing the raw image. This file can then be mounted using the standard loopback device.

```
~/pyflag# ./launch.sh \
./utilities/fuse_loopback_subsystem.py /tmp/mnt/ \
 -i raid -header 0 -blocksize 4k -slots 3 -map \
0.1.P.P.2.3.5.P.4 -offset 63s -filename d?.dd

~/pyflag# mount -oloop /tmp/mnt/mountme /mnt/data/
```

How does this information affect me?

- **Traditionally RAID acquisition was difficult and slow:**
  - Now we can acquire individual disks in parallel:
    - For example 3TB Apple X-Raid has 14 disks of 250GB each
    - Assuming about 2GB/Min acquisition rate it would take around 4 hours to acquire each disk.
    - Acquiring linearly would take approximately 36 hours and would require a 3.5 TB server to place the image into.
  - Traditional methods involve copying over USB/FireWire or Ethernet – both would achieve far less than 2GB/Min typically.

Conclusions

- **Acquiring Large RAID arrays is now practical and possible**
- **Automated techniques were presented for recovering RAID logical images**
- **Data recovery of RAID arrays is possible.**