The submission contains four raw (dd) image files of the USB flash disk «Transcend JF V10 / 1GB, D33193», two packet capture (pcap) files and four log files. The disk is non-partitioned and contains no file systems; it contains many non-deterministic sectors (each sector contains 512 bytes).

Namely, each sector that doesn't belong to a written block of flash memory cells contains non-deterministic data (instead of null bytes, as many forensic examiners tend to expect). The disk does function properly though. Several tests show that writing to a sector turns its contents to deterministic state (i.e. you will read exactly what you wrote).

Days were spent to understand why there are non-deterministic blocks of data. The study showed that each non-deterministic sector represents the contents of the SCSI READ(10) command related to reading that sector. In other words, when the disk receives SCSI READ command that covers non-written sectors it simply sends the contents of the command back to the host, and these contents appear as sector data to an operating system.

In the experiment two raw images of the USB flash disk were acquired on a Linux host using dc3dd (these image files together with corresponding dc3dd log files can be found in «linux-dc3dd/»), and two other raw images were acquired on a Windows 7 host using FTK Imager (image files and log files are located in «windows7-ftkimager/»); all images have different hash values. Windows host was also running capture software to intercept all USB commands and replies, this data was written to pcap files named «usb-1» and «usb-2» (for the first and the second acquisition accordingly). There were no writes to the disk during or between acquisitions. The disk was disconnected between acquisitions on a Windows host: this was done to assign a new tag to the command blocks of all SCSI READ(10) commands going to the disk (unlike Linux, Windows uses the same tag in the command block of all SCSI READ(10) commands, the tag seems to be generated randomly when a disk is connected via USB; Linux, conversely, assigns new tag to every command block of SCSI READ(10) command); otherwise, two images would have the same hash value on a Windows host (results of hashing the disk twice without reconnecting it are shown on the screenshot located at «windows7-ftkimager/ftk-imager-screenshot.png»).

Let's look at the sector #100005 in four images acquired (dd options: skip=100004 count=1).

«linux-dc3dd/flash-firstrun.dd» has the following data:

```
00000000  55 53 42 43 94 06 00 00  00 80 00 00 80 00 0a 28  |USBC...........(|
00000010  00 00 01 86 80 00 00 40  00 00 00 00 00 00 00 60  |.......@.......`|
00000020  00 60 ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |.`..............|
00000030  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000200
```

«linux-dc3dd/flash-secondrun.dd» has the following data:

```
00000000  55 53 42 43 00 7f 00 00  00 80 00 00 80 00 0a 28  |USBC...........(|
00000010  00 00 01 86 80 00 00 40  00 00 00 00 00 00 00 44  |.......@.......D|
00000020  00 44 ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |.D..............|
00000030  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000200
```

«windows7-ftkimager/flash-firstrun.001» has the following data:

```
00000000  55 53 42 43 d8 b1 6b 91  00 40 00 00 80 00 0a 28  |USBC..k..@.....(|
00000010  00 00 01 86 a0 00 00 20  00 00 00 00 00 00 00 5a  |....... .......Z|
00000020  00 5a ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |.Z..............|
00000030  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000200
```

«windows7-ftkimager/flash-secondrun.001» has the following data:

```
00000000  55 53 42 43 20 6a 7f 83  00 40 00 00 80 00 0a 28  |USBC j...@.....(|
00000010  00 00 01 86 a0 00 00 20  00 00 00 00 00 00 00 79  |....... .......y|
00000020  00 79 ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |.y..............|
00000030  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000200
```

As you can see, sectors are slightly different. Now let's dissect the data in the first hexadecimal dump (using various colors to highlight the bytes):

```
00000000  55 53 42 43 94 06 00 00  00 80 00 00 80 00 0a 28  |USBC...........(|
00000010  00 00 01 86 80 00 00 40  00 00 00 00 00 00 00 60  |.......@.......`|
00000020  00 60 ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |.`..............|
00000030  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000200
```

Letters «USBC» point us to the USB command block, which starts with four-byte signature «USBC». The structure of the USB command block is presented below (taken from Linux kernel source code):

```
struct bulk_cb_wrap {
        __le32  Signature;               /* contains 'USBC' */
        __u32   Tag;                     /* unique per command id */
        __le32  DataTransferLength;      /* size of data */
        __u8    Flags;                   /* direction in bit 0 */
        __u8    Lun;                     /* LUN normally 0 */
        __u8    Length;                  /* of of the CDB */
        __u8    CDB[16];                 /* max command */
};
```

The CDB field contains an actual command transmitted. The first byte of the CDB field is 0x28, which refers us to the SCSI READ(10) command, which operational code is 0x28 (see Table 85 in the «SCSI Commands Reference Manual» by Seagate: www.seagate.com/staticfiles/support/disc/manuals/scsi/100293068a.pdf). SCSI READ(10) command is exactly ten bytes in length (not counting previous USB command block header).

```
struct read_10 {
        __u8     opCode;          /* operational code (28h) */
        __u8     Flags;           /* various flags */
        __u32    LBA;             /* logical block address (MSB first) */
        __u8     Group;           /* group number */
        __u16    TransferLength;  /* transfer length (MSB first) */
        __u8     Control;         /* control byte */
};
```

The contents of SCSI READ(10) command are: logical block address is 18680 in hexadecimal, or 99968 in decimal; transfer length is 40 logical blocks in hexadecimal, or 64 in decimal. Note that we were analyzing sector #100005, which is between 99968 and 100032 (99968+64). Now let's check what data is present in the sectors #99967 till #100032 (one-liner for bash: «for i in `seq 99967 100032`; do echo -n "$i: "; dd if=flash-firstrun.dd skip=$i count=1 2> /dev/null | md5sum; done»): sectors #99968 till #100031 have the same data as sector #99968; sector #99967 differs from them, as well as sector #100032. The conclusion is that sectors #99968 till #100031 have non-deterministic data, which represents the contents of the SCSI READ(10) command used to read that sector range.

A program was written to study all non-deterministic sectors the same way as described above, and the results are the same — every non-deterministic sector contains a corresponding SCSI READ(10) command.

Related links

1. http://www.forensicfocus.com/index.php?name=Content&pid=366 (Flash drives and acquisition by Dominik Weber)
2. http://www.forensicfocus.com/index.php?name=Forums&file=viewtopic&t=4707 (FAT32 strangeness by «Fab4»)