A Comprehensive Black-box Methodology for Testing the Forensic Characteristics of Solid-state Drives

Gabriele Bonetti, Marco Viglione, Alessandro Frossi, Federico Maggi, Stefano Zanero
DEIB, Politecnico di Milano
{gabriele.bonetti,marco.viglione}@mail.polimi.it, {frossi,fmaggi,zanero}@elet.polimi.it

ABSTRACT

Solid-state drives (SSDs) are inherently different from traditional drives, as they incorporate data-optimization mechanisms to overcome their limitations (such as a limited number of program-erase cycles, or the need of blanking a block before writing). The most common optimizations are wear leveling, trimming, compression, and garbage collection, which operate transparently to the host OS and, in certain cases, even when the disks are disconnected from a computer (but still powered up). In simple words, SSD controllers are designed to hide these internals completely, rendering them inaccessible if not through direct acquisition of the memory cells.

These optimizations have a significant impact on the forensic analysis of SSDs. The main cause is that memory cells could be preemptively blanked, whereas a traditional drive sector would need to be explicitly rewritten to physically wipe off the data. Unfortunately, the existing literature on this subject is sparse and the conclusions are seemingly contradictory.

In this paper we propose a generic, practical, test-driven methodology that guides researchers and forensics analysts through a series of steps that assess the "forensic friendliness" of a SSD. Given a drive of the same brand and model of the one under analysis, our methodology produces a decision that helps an analyst to determine whether or not an expensive direct acquisition of the memory cells is worth the effort, because the extreme optimizations may have rendered the data unreadable or useless. We apply our methodology to three SSDs produced by top vendors (Samsung, Corsair, and Crucial), and provide a detailed description of how each step should be conducted.

1. INTRODUCTION

Solid-state drives (SSDs) have reached remarkable popularity nowadays, as their increasing capacity and affordable prices made them a good alternative to standard, platter-based hard drives (HDD, from hereinafter) [10]. SSDs offer the flexibility and compatibility of traditional drives, long with the shock-resistance ensured by the lack of mechanical components typical of flash drives, and the speed offered by flash memories.

ACSAC '13 New Orleans, Louisiana, USA

SSD have a shorter lifespan than HDDs. NAND-based flash chips, in fact, have a physical limit of around 10,000 program-erase cycles. When approaching and surpassing this limit, NAND floating gates exhibit problems in retaining their charge and, if not constantly refreshed, they lose their content. This means that keeping an SSD without power for a couple of days may lead to data loss. While 10,000 cycles may seem a very high number, it is a rather low lifespan for hard drives. Another limitation is that if blocks need to be rewritten, they must be blanked first, causing an extra overhead. This issue is further exacerbated in SSD drives, where the smallest addressable unit is a 16 to 512k block. SSD vendors have developed specific techniques such as write caching, trimming, garbage collection and compression, which aim at reducing the actual number of physical program-erase cycles. With these optimizations, SSD controllers are much more active than HDDs controllers, which let write and read requests pass through.

As a consequence, existing and widely-adopted forensic data-acquisition and analysis procedures may not be completely suitable for SSDs (e.g., the hash of an SSD may not be "stable" over time, as obsolete data may be automatically wiped by internal optimizations). The only viable option is a white-box acquisition that bypasses the controller and reads the content of the NAND chips. Unfortunately, as explained in §2.1, a white-box acquisition is expensive, not always feasible, can possibly disrupt the drive, and may lead to the conclusion that data is lost or damaged. In this regard, it would be useful to have a simple and affordable (black-box) triage procedure to decide whether a white-box analysis may produce some usable outcome on given the SSD brand, model and release, and OS.

In this paper we propose a generalized, practical analysis methodology to selectively address the peculiarities of SSDs that may impact forensic acquisition and reconstruction. Our methodology is a test-driven workflow that guides the forensic analyst through a series of experiments. The goal of each experiment is to assess how the controller logic behaves under different conditions and provide the analyst with useful insights on how the SSD under examination works and what are the optimizations adopted. Given the SSD model, brand and release, and the OS (if any) used on that SSD, our workflow provides (i) insights on the potential impacts of such optimizations on the results of standard forensic tools, and (ii) a practical decision framework to determine the expected success rate of retrieving lost data through white-box analysis.

As our methodology is black-box, it is transparently applicable to any SSD brand and model without any modification. Throughout the paper, we show this by applying our workflow on three SSDs of different vendors, each with a different controller, chosen because they are the most used ones: we cover this way a very vast variety of devices on the market and can analyze their peculiar behaviors, directly tied to the controller they are built with. Regardless of

the specific experiments that we carry out for the sole purpose of demonstrating the practicality of our workflow, we show that a forensic analyst can use our tests to assess whether a certain feature is implemented in an arbitrary SSD. Based on the result of each assessment, for which we provide real-world results, the analyst can consider the possibility of proceeding with a white-box analysis (for instance, if the TRIM or wear leveling are implemented).

In summary, this paper makes the following contributions:

- We propose a test-driven, black-box methodology to determine whether a SSD implements trimming, garbage collection, compression and wear leveling.
- We show how our the outcome of our methodology guides the practitioners in understanding how they impact their chances of data retrieval using traditional black-box, or expensive white-box analysis techniques.
- We show our methodology by applying it on three popular SSD brands and models, and detail precisely how each step is conducted and how the results of each step are interpreted.

2. BACKGROUND AND MOTIVATION

SSDs employ a complex architecture, with many hardware and software layers between the physical memory packs and the external interface to the computer. These layers, merged in the flash translation layer (*FTL*) [12], are in charge of reading and writing data on the ATA channel on one side and on the memory chips on the other side, as well as to compress, encrypt or move data blocks to perform optimizations. In HDDs the OS has direct access to the data contained on platters, and the controller is limited to moving the magnetic head and read or write data. Instead the FTL of SSDs performs much more complex functions: It translates logical block addresses (LBA) as requested by the OS into the respective physical block addresses (PBA) on memory chips. The underlying mapping is completely transparent and can be modified by the FTL at any time for any reason. The need for mechanisms such as the FTL has been studied extensively by Templeman and Kapadia [16], who show that the likelihood of a cell wearing (i.e., losing its ability of retaining data) and its maximum lifespan. They show that the endurance of memories greatly varies among the vendors and chip models, and that premature decay is caused by stressing cells with continuous writes.

2.1 White-box Forensics Analysis

The action of the FTL is transparent to software and to the host OS: To the best of current knowledge, there is no way to bypass the FTL via software, and explore the raw content of the memory chips. Hardware intervention, also known as *white-box* acquisition, is required. Breeuwsma et al. [7] showed that it is possible to acquire data from a flash memory chip in several ways. One option is to use flasher tools that interact with the chip directly via the pins of the board; other ways are the use of a JTAG port usually left by vendors on devices to bypass the controller or, in extreme cases, the physical extraction of the chip for dumping via standard readers.

Although a complete white-box analysis of a SSD is theoretically possible and in some cases feasible, it is also very difficult, time consuming and expensive, because it requires custom hardware. Creating custom hardware requires the forensics analyst to acquire specific skills, buy expensive equipment, and, once a successful acquisition is finally carried out, spend a significant amount of time to reverse the implementation of the SSD's controller policies. We attempted to read directly from the memory chips using non-expensive (i.e, tenths of US dollars) clips[1] but we obtained a fragmented, incomplete raw file. The techniques developed for small memories could be ported to SSDs, but results are not guaranteed. As a matter of fact, the applicability of white-box techniques highly depends on the disks architecture and hardware design. We performed an exploratory experiment with three drives (Samsung, Corsair, and Crucial) and limited hardware resources and found it very hard even to access the chips on the board, without disrupting them, or find accessible JTAG ports: Understandably, vendors tend to protect their intellectual property (i.e., the FTL algorithms) by not allowing this kind of access to the hardware. Last, but not least, white-box approaches must deal with data-compression features: The sole knowledge of the compression algorithm is not sufficient. Indeed, the analyst would need to know the compression algorithm *and* the data allocation policy (i.e., how bytes are spread over the memory chips), which is definitely a protected (or at least, not public) information.

2.2 Black-box Forensics Analysis

Differently from white-box approaches, *black-box* approaches read data as presented to the ATA interface by the SSD controller.

Bell and Boddington [5] analyzed the file recovery rate on SSDs versus HDDs during a standard, black-box forensic acquisition. When issued a quick format command, the SSD used in the experiment wiped the entire content (i.e., text files) irrecoverably in a matter of minutes. They confirmed this result with a write blocker (i.e., re-attaching the SSD to a write blocker after the quick format), showing that this deletion did not happen as a result of commands issued by the host or its OS: SSDs can indeed schedule and perform their own write operations. This work provided one of the first hypothesis on how garbage-collection algorithms work, stating that some of them (primarily Samsung) are capable of "looking at the used/unused aspects of an NTFS filesystem by examining the free space bitmap". The authors hypothesized that these controllers may be file-system aware, and need no OS intervention to blank unused blocks. This poses major issues, rendering traditional forensic methodologies (such as the use of write blockers) insufficient to preserve the digital evidence. However, as we report in §4.2 we were unable to replicate their experiment, even using the same OS, scripts, drive (including firmwares and versions) and working conditions. Even with the authors' help, we were unable to find the reason for this difference.

From the one hand, treating SSDs just like HDDs with black-box tools ensures partial observability over the controller's behavior. On the other hand, black-box approaches are more practical and convenient than white-box approaches. Notably, they are less obtrusive and expensive. Unfortunately, to the best of our knowledge, there is no SSD alternative, scientifically-tested black-box methodology that supports the forensic analyst in evaluating the soundness of the results produced by existing tools.

2.3 Challenges and Goals

When applied to SSDs, black-box and white-box approaches have symmetric advantages and drawbacks: the former fails when a drive performs internal optimizations silently, whereas the latter fails on proprietary hardware, which is difficult to manipulate and access, or when data is compressed or encrypted. Indeed, no single hardware tool or methodology can help with every SSD drive, since each of them has a different architecture, different chips positioning, and many other details that make "generic" hardware tools impossible to build. In addition, SSDs use different flash memory chips; these

[1] We used the TSOP NAND clip socket, available online for 29USD.

have very different working parameters. In some cases, it may be physically impossible to connect to the memory chips because of the way they are soldered to the board; as a consequence the chip often needs to be removed, thus potentially damaging the drive or destroying the evidence. SSD controllers are specifically built to reach high throughputs by leveraging parallel reads and writes; custom forensic hardware is much slower and can read only one chip at a time, making the dump (and the reconstruction) of an entire drive a very long process. Moreover, data compression or proprietary encryption may easily disrupt the entire white-box analysis, and also to the capabilities of certain controllers to wipe obsolete blocks and make it impossible to recover deleted files.

On the other hand, black-box approaches are less precise. However, being independent from the proprietary hardware, they can be easily generalized. Our key observation is that a black-box triage is a mandatory prerequisite before committing resources to a challenging, costly and potentially fruitless white-box analysis.

In summary, as every drive and every controller behave differently one from the other, we focus on providing a general methodology to perform forensically sound tests and determine how the FTL of a given SSD affects the results that standard forensic techniques may yield. Therefore, the main goal of our work is to devise an analysis methodology that advances the state of the art for its generality, and hopefully offers a useful reference for forensic practitioners and researchers. The second goal is to provide pure black-box techniques to "estimate" the likelihood of retrieving additional data through a white-box effort, allowing forensic experts to triage evidence and avoid wasting resources. Last, we strive to replicate and validate the experiments described in the literature to take into account the previous conclusions in our methodology.

3. METHODOLOGY OVERVIEW

The input of our methodology is an SSD of the same brand and model of the one under examination. The first step is to conduct a series of tests that determines whether that SSD implements certain features (regardless of what the vendors state), how fast and aggressive they are with respect to stored data and how they would influence forensic data reconstruction. The first step covers the following aspects:

TRIM This functionality mitigates the known limitation of SSDs that requires any block to be blanked before it can be rewritten. The trimming function erases data blocks that have been marked as "deleted" by the OS. Trimming has a negative impact on forensic analysis. Indeed, on-disk data persistence after deletion is no longer guaranteed: Once a block is marked as free by the OS, the controller decides when to blank it according to its policies. As noted in [5], this can occur regardless of data connection between the SSD and a host computer (e.g., during an acquisition, even when write blockers are used). Our methodology can determine the percentage of blocks that get erased and how fast this happens (§4.1).

Garbage collection (GC) This is a functionality that SSD vendors often list as one of the most useful and interesting feature, capable of greatly improving the drive's performance. However, as explained in §3.1 and 4.2, from our investigation we conclude that it is very hard even to define the concept of garbage collection. Bell and Boddington [5] hypothesize that GC works by making the controller somehow aware of the filesystem, and able to infer on its own which blocks are obsolete by monitoring the file-allocation table. If this were the case, GC would bias forensic acquisitions significantly. GC is not triggered by the OS; consequently, data could be erased whenever the disk is powered on, even if no write commands are issued, thus rendering write blockers and other precautions useless. Therefore, it is important to know whether a SSD implements GC. Our methodology can determine whether GC is active.

SSD	WL	TRIM	GC	Compression
Corsair F60	✓	✓	✓	✓
Samsung S470	✓	✓	✓	
Crucial M4	✓	✓		

Table 1: SSD features as reported by vendors.

Erasing patterns Some SSDs show peculiar behaviors when using TRIM: They do not erase all the marked blocks but rather a subset of them based on some drive's internals. Our methodology explores this behavior to characterize the erasing patterns (§4.3).

Compression Some drives transparently compress data to use less blocks and reduce cell wearing. Compression poses no direct challenges in black-box forensics acquisitions, whereas it makes white-box analysis useless, as the data read directly from the chips would be unusable, unless the compression algorithm were known or reverse engineered. Therefore, in our methodology we included a step that can verify whether compression is active.

Wear leveling (WL) This functionality reduces the usage of flash cells by spreading their consumption as evenly as possible across the drive. To do so, the FTL may activate wear leveling if certain blocks have reached an excessive number of writes compared to the rest of the disk. The easiest and least expensive wear-leveling implementation allows the FTL to write the new block on another, less-used portion of the disk, and updates the internal file mapping table that the FTL maintains [2]. Alternatively, vendors could provide the disk with extra physical space, so that new blocks can be written on brand new memory cells. This second technique is used, for example, by the Corsair F60 SSD, which has a total of 64GB flash memory but allows to address only 60GB at a time; the drive itself is more expensive for the vendor, but if the wear leveling functionality is correctly implemented it grants a much longer lifespan. Although wear leveling is quite standard in modern drives, it is very useful to know whether the SSD implementation masks the effect of the so-called "write amplification" (see §4.5), which is a direct consequence of the wear leveling.

Files recoverability Even though blocks may not be erased, they might be changed or partially moved, making it impossible to retrieve them through carving. This test checks how many deleted files can be retrieved from the drive (§4.6).

Given the outcome of these tests, the second step is to interpret the results and provide a ranking of a drive in terms of its "forensic friendliness", as detailed in §5. Our methodology covers all known combinations of factors that may trigger each feature. In addition, our methodology is designed to avoid redundant tests, which would certainly not trigger any of the features. When feasible, we compare our results with the outcome of previous studies. In particular, we validate the experiments of Bell and Boddington [5] on garbage collection, which is a particularly controversial and ill-defined topic as detailed in §3.1.

3.1 Garbage Collector vs. Garbage Collection

The difference between garbage collector and garbage collection is a controversial concept that needs to be clarified before explaining our methodology. Many works in the literature treat these two features as being substantially the same and propose methods to trigger and reverse-engineer this functionality. They are, however, two different (logical) components of an SSD controller, which unfortunately share a very similar name causing considerable confusion.

For the purpose of our work, the garbage collect*or* is the *process* that deals with unused blocks—as a garbage collector does with unused variables in modern memory managers. The internals of SSD garbage collectors are not disclosed, and they may vary from drive to drive. However, it is known that the garbage collector is tightly tied to the TRIM functionality: among other capabilities, it has access to the "preemptive-erase table" (§4.1) filled by TRIM and takes care of physically wiping trimmed flocks. Additionally, the garbage collector helps wear leveling by moving around blocks whenever the wear factor of a cell is beyond a certain threshold.

On the other hand, vendors never disclosed any details about the garbage collect*ion functionality*, although there were some speculations about how it is supposed to work. The only known work is by Bell and Boddington [5], who—partially supported by Samsung—hinted that the garbage collection functionality allows the drive controller to introspect the file allocation table of known file systems (i.e., NTFS, ext3 and ext4) to autonomously decide which blocks can be safely wiped, without the OS intervention or the implementation of TRIM. As a matter of fact, in §4.2 we document our tests on the garbage collection functionality under these hypotheses to see whether the results from various drives could confirm its presence and behavior. Our experiment lead to results that contradict Bell and Boddington's work.

3.2 Write Caching in SSD Experiments

SSDs are equipped with a small amount of DRAM-based cache memory, which functions is to reduce the number of physical writes. This feature must be taken into account when performing experiments on SSDs, because its obvious side effects. Caching can be ignored in tests that use large files, as it is negligible with respect to the drive capacity and is therefore bypassed. However, it must be taken into account when performing fine-grained tests.

Biases introduced by caching were not addressed by Antonellis [4], who performed experiments by writing everyday-use graphic and text files on the SSD and then verified that they were completely unrecoverable after deleting them. Although in [4] there is no clear statement about th OS used, in 2008 no OS had TRIM support for SSD drives [3] and therefore the file deletion could not be a consequence of TRIM. The only explanation is that the files the author used were not big enough to completely fill the drive cache (usually around 512MB to 1GB) and were therefore never actually written on disk: they were simply erased from cache and no trace was left to allow a full or partial recovery. Also in [13] this effect is noticed: The percentage of recoverable blocks when using small files is considerably lower than the same percentage with big files, even under the same conditions and usage patterns. This can be explained by the fact that small files get usually stored in cache and not written to disk.

Our methodology requires that cache is disabled either from the OS (e.g., via the `hdparm -W 0` command) or by using very large files for write operations whenever possible, to fill up the cache and force the drive to physically write on flash cells.

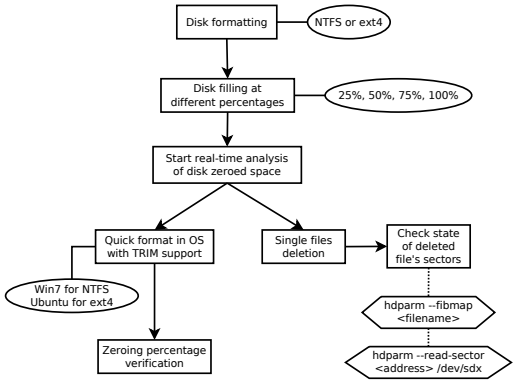


Figure 1: TRIM test flow.

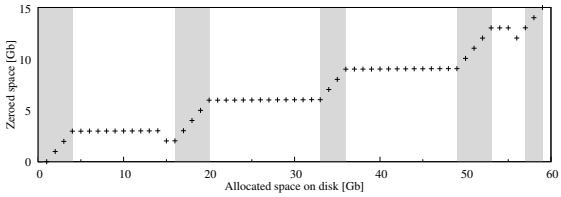


Figure 2: The amount of blocks erased by TRIM in our Corsair F60 disk depends on the amount of used space.

4. IMPLEMENTATION DETAILS

We apply the methodology described in this section on three SSDs, each with a different controller and combination of features, namely Corsair F60 (controller: SandForce SF-1200), a Samsung S470 MZ-5PA064A (controller: Samsung ARM base 3-core MAX) and a Crucial M4 (controller: Marvell 88SS9174-BLD2). As stated by the vendors, these drives implement wear leveling. Furthermore, the Corsair F60 performs data compression, whereas both Samsung S470 and Corsair F60 are said to implement garbage collection. Table 1 summarizes the functionalities according to the official specifications. Instead of presenting the results of our tests in a separate section, for the sake of practicality we explain the results immediately after the description of each step.

4.1 TRIM

Using trimming, whenever blocks are erased or moved on an SSD, they are added to a queue of blocks that should be blanked. This operation is lazily performed by the garbage collector process as soon as the disk is idle, making trimmed blocks ready to be written and ensuring balanced read-write speeds. Trimming is triggered by the OS, which informs the controller when certain blocks can be trimmed. We focused on Windows and Linux: Windows 7 and 8 (and Server 2008R2), and Linux from 2.6.28 on support trimming.

Methodology. Fig. 1 shows the steps required to determine whether and how an SSD implements trimming. Before start, the disk is wiped completely and the write cache is disabled or filled, as detailed in §3.2. Then, a stub filesystem is created and filled with random content (i.e., files) up to different percentages of their capacity: 25, 50, 75 and 100%. This is because certain controllers exhibit different trimming strategies depending on the available space. The tests described below are repeated for each percentage.

As both [13] and our experiments show that some TRIM implementations behave differently when dealing with (1) quick formats or (2) file deletions, we analyze both cases independently. When a (1) quick format command is issued, the OS will supposedly notify the SSD that the whole drive can be trimmed. The disk is left idle and the filesystem is checked for zeroed blocks. If the SSD

implements TRIM procedures, we expect to observe changes in the number of zeroed blocks; otherwise, no changes will happen. To check for zeroing, we use a sampling tool—as Bell and Boddington [5] did—which loops over the entire disk and samples 10Kb of data out of every 10MB. It then checks whether the sample is completely zeroed. Whenever a non-zeroed sample is encountered, the tool checks whether it is zeroed in subsequent loops. The sampling size was chosen empirically to find a good trade-off in terms of overhead and accuracy. The choice of the sampling size depends on the time that the analyst wants to spend on this test, and only affects the final decision. The test ends when the situation does not change within a timeout, which can be sometimes obtained by the vendor's documentation (i.e., the time between cycles of running of the garbage collector). If no documentation is available, we set a very long timeout (i.e., 24 hours, based on several trials that we run on all our disks). During our experiments, we found out that if the TRIM functionality is present and active, it triggers within 1 to 10 seconds. Similarly, (2) our workflow deletes single files from the filesystem and monitors their respective blocks. Depending on the file size, sampling may not be necessary in this case. The test ends, as before, when all the (remaining) portions of the erased file do not change within a timeout.

Results. We run this test on Windows and Linux, which both included stable support for TRIM.

On NTFS (Windows 7), Samsung S470 and Crucial M4's trimming was very aggressive in both quick format and file deletion: The disk was wiped in 10 seconds by the Samsung S470 controller; on the Crucial M4, wiping occurred even before notifying the OS. Similar results were obtained with file deletion: the sectors were completely wiped in 5 and 10 seconds respectively. The Corsair F60, instead, behaved differently. After issuing a quick format, only a small percentage of data was erased; when we repeated the test at different filling levels, we surprisingly found out that the fraction of erased blocks is someway proportional to the total used space, as shown in Fig. 2: There are some thresholds that define how much space must be trimmed depending on the used space. In particular, there are 5 ranges in which the amount of zeroed space increases linearly, whereas it remains constant at all the other filling values. The Corsair F60 behaved unexpectedly also when dealing with file deletions: Some files were wiped in at most 3 seconds after deletion, whereas some other files were not wiped at all and could be recovered easily, depending on their allocation. This discovery spurred an interesting study of the erasing patterns, which is explained separately in §4.3.

On ext4 (Ubuntu Linux 12.04) we obtained significantly different results. In the quick-format branch the outcome are similar across different disks: The entire content of the SSD was erased in about 15 seconds. This can be explained by the fact that for all the SSDs Linux used the same AHCI device driver. The single file deletion, instead, showed a different behavior. The Samsung S470 did not erase any block and all the files were completely recoverable. The Crucial M4 apparently did not erase any file, at least until the device was unmounted; at that point the blocks were erased. Apparently, the driver notifies a file deletion only when it becomes absolutely necessary to write data on disk (i.e., when the disk is unmounted or when the system is in idle state long enough to flush data on the non-volatile storage). The Corsair F60 showed none of the behaviors exhibited with NTFS: All the files were erased correctly. Supposedly, Windows drivers implement a different trimming policy, or the SandForce controller used by this SSD features NTFS-specific optimizations.

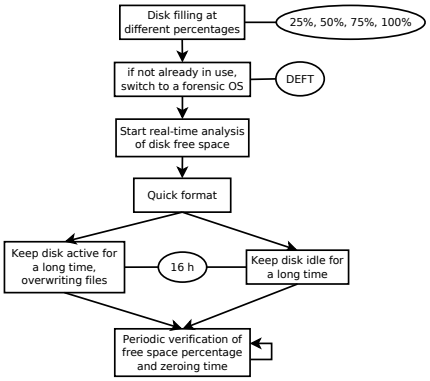


Figure 3: Garbage collection test flow.

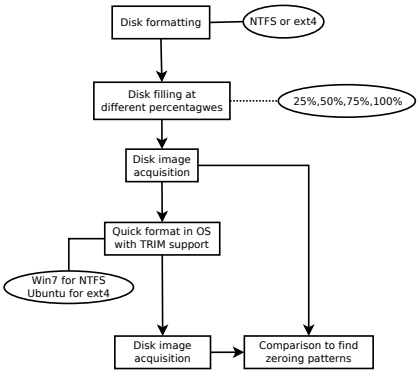


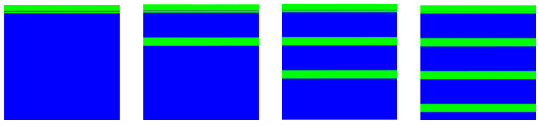
Figure 4: Erasing patterns test flow.

4.2 Garbage Collection

Methodology. The entire test must carried out on an OS and drivers that do not support TRIM, to avoid interferences between TRIM and GC, which effects are indistinguishable from a black-box viewpoint. Fig. 3 summarizes our test to determine whether GC is implemented and when it starts.

After the usual preliminary steps, the dummy filesystem is created and filled with random files. The content is not important, yet the size is, as small files are not physically written on disk. This is particularly important when write caching cannot be disabled reliably. Then, the same sampling procedure described in §4.1 is started, and a quick format is issued. As there is no reliable information regarding the triggering context and timeout, our methodology explores two different paths. First, the disk is kept idle to allow the triggering of the GC. Alternatively, the SSD is kept active by continuously overwriting existing files, adding no new content. Bell and Boddington found out that GC triggers in almost 3 minutes. Some non-authoritative sources, however, state that a reasonable timeout ranges between 3 to 12 hours. Our methodology proposes to wait up to 16 hours before concluding the experiment.

Results. Even hours after the default timeout, none of the SSDs performed GC. Since Samsung S470 and Corsair F60 were advertised as having GC capabilities, we devised a simple additional test to validate this result. This goal was to determine what percentage of non-random files can be recovered after a quick format. We filled each disk with the same JPEG image until there was no space left on the device, and formatted it on a TRIM-incompatible OS and let it idle. As shown in Table 5, even with simple tools (e.g., Scalpel), we recovered 100% of the files from both drives, confirming that no GC occurred. Note that we used a carver merely as a baseline for recoverability: Our approach is not meant to evaluate the performance of carvers in general.



(a) 10%. (b) 50% (c) 75% (d) 100%

Figure 5: Test results for erasing patterns test performed on Corsair F60 SSD: at different filling levels an increasing number of evenly-spaced stripes are visible. Green areas are zeroed by the controller, while blue areas remain unchanged. The non-erased blocks in the first stripe (a) contain the copy of the master file table and are therefore not zeroed.

4.3 Erasing patterns

As showed in §4.1, certain SSD controllers (e.g., Corsair F60 with NTFS) may exhibit unexpected trimming patterns. Therefore, we devised a workflow to further explore these cases and assess to what extent a forensic acquisition is affected by TRIM or GC.

Methodology. As shown in Fig. 4, after the preliminary steps the disk is filled with a dummy filesystem containing files with random data (the content is irrelevant as we are focusing on how the controller handles deletion). Then, a raw image of the disk is acquired with `dd` before issuing a quick format instruction. A second raw image is then acquired after a while (see the considerations in §4.1 on timeouts). Clearly, "raw" here refers to what the controller exposes to the OS. The obtained images are compared block-wise to highlight erased sections. This test must be ran at different disk filling levels, in case the controller behaves differently based on the amount of free space.

Results. We applied this test on the Corsair F60 SSD, which exhibited odd behavior in the TRIM test. We analyzed it at 10, 50, 75 and 100% of space used. At each level we analyzed the entire disk as explained above and created the maps shown in Fig. 5. Interestingly, we notice four stripes in predictable areas (green) where the files are surely going to be erased, whereas the rest of the disk (blue) is not modified even after file deletion. The small difference in the first stripe is due to the fact that the master file table is allocated within it, and this portion is not erased (Fig. 5(a)).

This result is consistent with the results described in §4.1. In particular, the first four linearly-increasing portions that appear in the TRIM test result of Fig. 1 correspond to the very same green areas highlighted with this erasing-pattern experiment. Therefore, the reaction of the controller when dealing with file deletion on NTFS is explainable by the very same green areas: If a file is allocated within the green stripes, it will surely be erased by TRIM, whereas files that fall outside the green areas are not trimmed.

We validated this result as follows. We formatted the drive and filled it with easily-recoverable files (i.e., JPEG image files, as the JPEG header is easy to match with carvers). Then, we selectively deleted the files allocated inside or outside the green stripes and, after acquiring the entire disk image, tried to recover them, and to map their (known) position against the stripes position. Table 2 shows that only 0.34% of the files within the erased stripes are recovered, whereas this percentage reaches 99% for files allocated entirely outside the green areas, thus confirming the results of Fig. 5.

Position	Written	Recovered	%
Within erased stripes	29,545	1	0.34 %
Outside erased stripes	71,610	71,607	99 %

Table 2: File-recoverability test for Corsair F60 SSD: Only one of the files that were written within the erased areas could be recovered, whereas 99% of those outside those bounds could be retrieved with standard tools.

Figure 6: Compression test flow.

4.4 Compression

Methodology. The key intuition behind our test is that the overhead due to hardware compression is negligible in terms of time. Thus, it will take considerably less time to physically write a compressed file with respect to an uncompressed one. However, this could not be the case if the controller actually goes back and compresses the files afterward as a background task.

As shown in Table 3, compression algorithms yield the best results with low-entropy files, whereas are not very effective on high-entropy data. We therefore created two different files with very different levels of entropy: `/dev/zero` and `/dev/urandom`. The methodology is summarized in Fig. 6. After creating the two files (10Gb each, to bypass write caching), we monitor via `iostat` the time spent in transfer and the throughput: A high throughput indicates compression, as less data is physically written on disk.

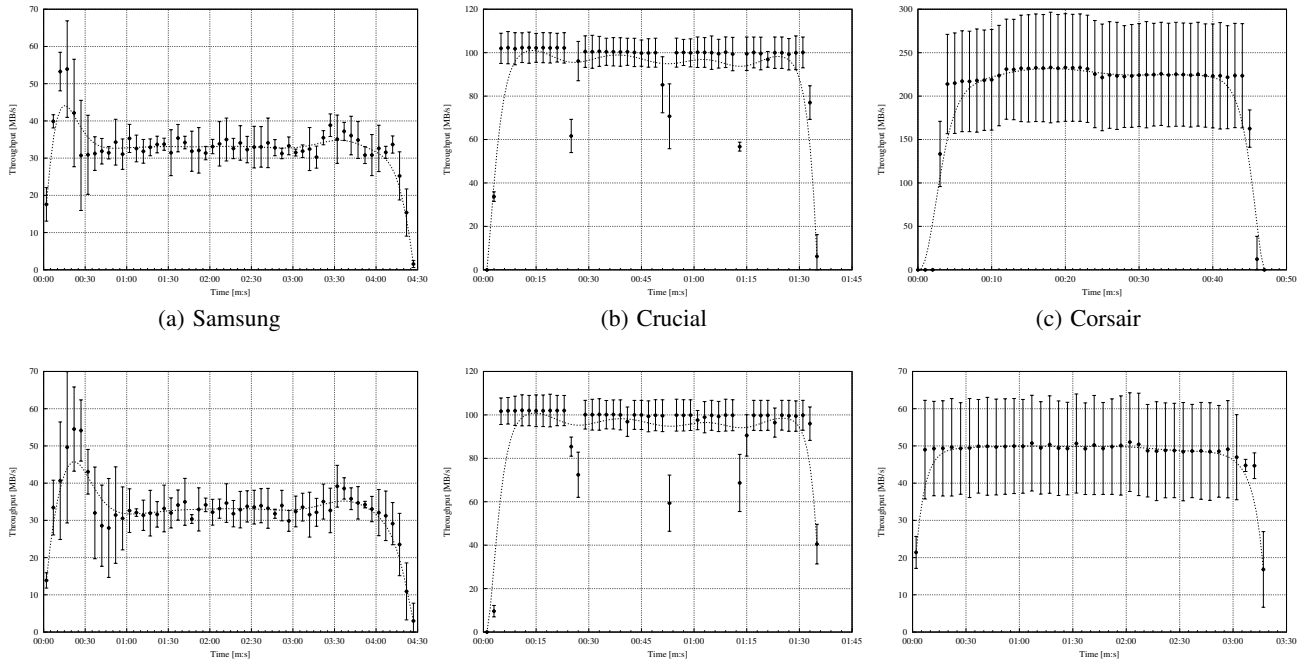
Results. As Fig. 7(a) shows, both file transfers in the Samsung S470 took almost the same time, showing no sign of compression.

The Crucial M4 test showed in Fig. 7(b) yielded the same results, even if with different values. This drive exhibits systematic performance glitches at the same points in time for each run. This happens almost every 25 seconds, regardless of the file's size and transfer time, and does not happen with other drives under the same conditions. We transferred files of different sizes and under different conditions (i.e., computer, source, OS) but obtained consistent results. We can speculate that these glitches are due to some computations performed by the controller at regular time intervals.

The Corsair F60 is the only one advertised as having compression capabilities. Indeed, as shown in Fig. 7(c), it behaves in a very different fashion: The transfer time for compressible files is about one third files than for incompressible files. Therefore, we can infer that the actual amount of data physically written on disk is considerably lower, meaning that the controller compresses it transparently.

File	gzip	7zip	bz2	Entropy
`/dev/zero`	1,030	7,086	1,420,763	0.0
`/dev/urandom`	1.0	0.99	0.99	0.99

Table 3: Compression ratio of the files used for compression test: files with high (Shannon's) entropy are difficult to compress and therefore result in more data to be written on disk.



(a) Samsung (b) Crucial (c) Corsair

Figure 7: Mean and variance of the sampled throughput among 15 repeated transfers of 10GB low and high-entropy files (top and bottom row, respectively). For (a) and (b) low and high-entropy file transfers have almost the same shape and duration, showing that the controller does not perform any kind of optimization (i.e., compression) on data before writing it. On the other hand, in (c) throughput with low-entropy files is considerably higher and the entire file transfer takes less than 1/3 than the high-entropy files transfer. This result confirms that less data had to be physically written on disk, which means that compression was indeed performed by Corsair drives.

All graphs show an initial transitory phase with a very high transfer rate. However, this is the effect of write caching on the very first megabytes of the file being sent to the disk. Disabling write caching via software drivers, as explained in §3.2, does not always succeed. The only reliable way to bypass caching was to use large files. Nevertheless, the effect of caching does not affect our results.

4.5 Wear Leveling

Wear leveling is quite common, although none of the examined vendors clearly states what happens to the old versions of the same file (i.e., how write amplification [11] is treated). From a black-box viewpoint there are two possible situations. One alternative is that old blocks are not erased and remain where they were: in this case a carver may be able to extract many different versions of the same file, representing a clear snapshot of the data at a given point in time. Alternatively, the old data may be erased, moved out of the addressable space or simply masked by the controller, which in this case would tell the OS that no data is present (virtually zeroed block) where obsolete data actually is. Unfortunately, if we get no data from the disk, there is no way (with a black-box approach such as ours) to determine which of these is the case.

Another detail to take into account when dealing with wear leveling is that vendors do not explicitly reveal the conditions under which the functionality is triggered. From the available information and previous work ([1, 2, 9]) it appears that two conditions must hold: there should be enough free blocks with a lower write count than the one that is being overwritten, and there must be a certain difference between the writes of the used block and the ones of the new destination block. Since the precise values depend on the vendors and are not publicly available, we erred on the side of caution and left, in our experiments, at least 25% of the disk capacity free

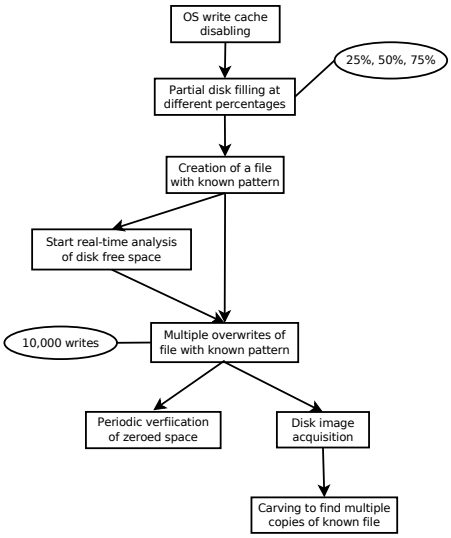


Figure 8: Wear leveling test flow.

and overwrote the same blocks more than 10,000 times to cover whatever write cycles gap may be present.

Methodology. Our test is not aimed at determining if an SSD implements a wear leveling feature, since this is pretty much standard nowadays. From the forensic viewpoint, what matters is if wear leveling can be leveraged via black-box analysis to recover data. If a drive has no wear leveling capabilities, or if write amplification is completely masked by the FTL, the end result is that nothing is lost and nothing is gained because of wear leveling.

Our test flow is shown in Fig. 8. An important preliminary step is to disable the OS and disk write caching, as it poses problems:

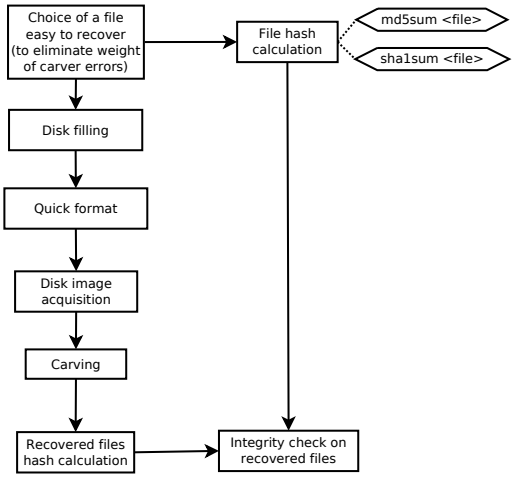


Figure 9: Files recoverability test flow.

The entire test flow requires the continuous re-writing of the same files, and it is extremely important that these write operations are physically sent to the disk.

Then disk is filled up to 75% with a dummy filesystem. Since wear leveling is internal, the file or filesystem type has no impact, so we choose files with known patterns (to ease carving operations afterward), and an ext4 filesystem under Ubuntu Linux.

At this point files are overwritten with new data (of exactly the same size) a total of 10,000 times while keeping zeroed space on disk monitored. If, at any time, zeroed space diminishes, it means that the controller wrote new data somewhere else, using less used blocks but leaving the old ones intact. Notice that remnant data could be garbage or unusable. For this reason, and only if the zeroed-space check gives positive results, it is advisable to perform a full disk acquisition with subsequent carving to determine if different versions of the same files are effectively recoverable. As said, in the very likely case that both checks yield negative results, we cannot know what the controller is really doing; what we know is just that a standard forensic procedure will not be impacted by wear leveling.

Results. We ran our test on all the disks in our possession. As expected, it is very unlikely to find a drive exposing multiple copies of the same files for a black-box analysis. Our results confirmed our expectation: both checks yield negative results. We know, however, that all of the drives actually implement wear leveling capabilities, as stated by their vendors. We can therefore only assume that the effect of write amplification are completely masked by the controller, which does not expose any internals.

SSD	FS	Written	Recovered	%
Samsung	**NTFS**	112,790	0	0 %
	ext4	110,322	0	0 %
Corsair	**NTFS**	101,155	71,607	70.79 %
	ext4	99,475	0	0 %
Crucial	**NTFS**	112,192	0	0 %
	ext4	110,124	0	0 %

Table 4: Files recoverability test results: the drives implementing an aggressive version of TRIM (Samsung S470 on NTFS and Crucial M4), did not allow the recovery of any file after a format procedure. The Corsair F60 on NTFS, as expected, has a non-null recovery rate due to the erasing pattern its TRIM implementation exposes. On ext4, however, this same disk allowed the recovery of 0 out of 99,475 files.

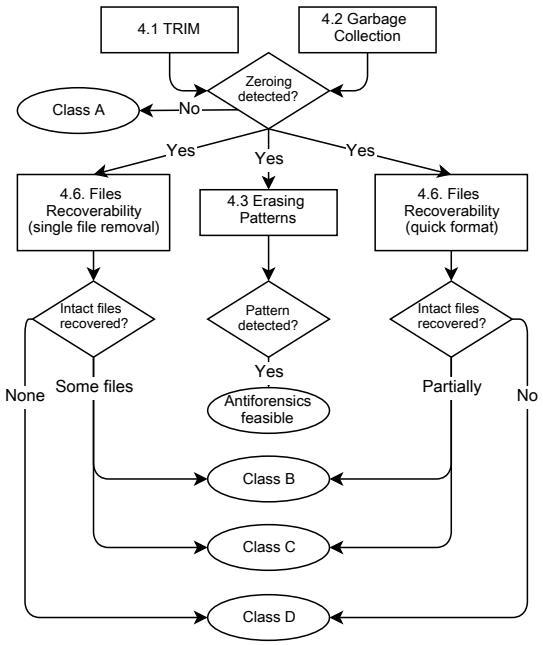


Figure 10: Use case workflow for assessing the forensic friendliness of a SSD.

4.6 Files Recoverability

The tests described so far are all aimed at determining if some functionalities implemented by a given SSD are forensically disruptive, to ultimately allow a forensic analyst to assess whether some data is still retrievable. What usually interests the forensic analyst most, however, is being able to access and retrieve files on an SSD much in the same way as on a traditional HDD.

The tests that we propose in this section determines how much an SSD behaves similarly to a HDD from a data-recoverability viewpoint. In HDDs, recoverability is affected by the filesystem policies on overwriting previous data. In SSDs, in addition to this, trimming, garbage collection, and the other unexpected controller behavior described so far negatively impact the recoverability.

Methodology. The test flow is shown in Fig. 9. The drive is first initialized with a dummy filesystem and filled with "carver-friendly" files: In our case, we wrote JPEG files of around 500k each, and then quick formatted the drive. After the usual 24 hours timeout, we used Scalpel to attempt a file recovery.

Results. We ran our experiment on all our disks with a NTFS filesystem with enough copies of the same JPEG image to fill the entire drive. As summarized in Table 4, both the Crucial M4 and the Samsung S470 have a zero recovery rate, which means that the TRIM functionality tested in §4.1 actually works and erases all of the deleted files.

The Corsair F60 behaves differently, as shown in §4.3: 71,607 files out of the 101,155 were recovered, totaling a 70.79% recovery rate on NTFS. Curiously, all the files that were only partially recovered—or not recovered at all—were all contiguous in small chunks. On Ext4, instead, TRIM did not allow the recovery of any file.

SSD	Written	Recovered	%
Samsung	112,790	112,790	100 %
Corsair	101,155	101,155	100 %

Table 5: Files recoverability without TRIM on Samsung S470 and Corsair F60 drives.

5. USE CASE: RANKING DRIVES

Although proposing a comprehensive and accurate classification of SSDs goes beyond the scope of this paper, we show how our methodology can be applied to indicate "forensic friendliness" of an SSD. We consider the outtput of the TRIM, GC and File Recoverability tests. We follow the workflow exemplified in Fig. 10.

- **A. Platter-disk Equivalent.** The SSD behaves as a HDD. Standard forensics tools are expected to work as usual. SSDs in this class present no disruptive behaviors (e.g., TRIM, GC).
- **B. High Recoverability.** TRIM and other wiping functionalities are implemented but they are not very aggressive: an HDD-equivalent recovery is expected.
- **C. Low Recoverability.** SSD's functionalities are quite aggressive and succeed in deleting or masking most of the deleted data that could have been recovered from a HDD. It is, however, still possible to achieve some results with standard tools.
- **D. Complete Wiping.** No deleted data can be recovered using standard black-box tools. White-box analysis may be a solution but it is not guaranteed to yield acceptable results. This is the worst possible case when performing a forensic analysis on a SSD.

Applying this method to our drives we obtained the following classification. Our Crucial M4 implements a very effective TRIM functionality with any filesystem which directly makes the recoverability test yield a 0% rate so, even though the garbage collector does not trigger, the associated class is *D. Complete wiping*: this drive is very likely to make recovery of deleted files impossible. The same happens on the Samsung S470 with both NTFS and ext4 filesystems and on the Corsair F60 with ext4. Corsair F60 SSD with NTFS, instead, present only a partially working TRIM implementation, which allows the recovery of almost 71% of deleted files; this combination between drive and filesystem is therefore associated to a *B. High recoverability* class.

6. LIMITATIONS

Although our blackbox workflow and experimental results are far more complete than what has been proposed in previous work, there are some limitations that is important to be aware of.

First, each SSD comes with its own firmware version, which basically embeds (part of) the FTL logic. As such, it determines the SSD characteristics and, therefore, its forensics "friendliness" with respect to the features tested by our workflow. We do not consider changing the firmware during our tests for several reasons. First, not all SSD vendors release firmware upgrades. Secondly, and most importantly, firmware upgrades are often one-way procedures; this affects the repeatability of the experiments in a scalable way (i.e., the only way to downgrade a firmware to a previous version would consist in buying another SSD, provided that the old version of the firmware is still on the market).

Second, the triggering of the TRIM depends on the specific combination of OS, filesystem type, device driver, and AHCI commands implemented. The current version of our workflow explores the OS and filesystem type. For instance we have shown in our experiments how the Corsair behaves differently under Windows (NTFS) and Linux (ext4). In a similar vein, other variables such as the device driver and AHCI commands can be considered. The reason why we have not included these variables into our methodology is because they vary significantly from product to product, whereas our main goal was to provide a generalized testing workflow.

Last, an intrinsic limitation of our approach is that the forensics examiner needs to know the OS version before performing an investigation. The availability of is this contextual information varies from case to case and it is hard to find previous data about this (due to the nature of the forensic investigations). Therefore we cannot make any strong statement on how likely it is to have such an information available.

7. RELATED WORK

In this section, we overview other relevant researches in the area of SSD forensics, in addition to the works presented in §2.

7.1 White-box Forensics Analysis

Similarly to Breeuwsma et al. [7], Skorobogatov [15] also addressed data acquisition from flash memory chips, but at a lower level. His technique, however, is not suitable for forensic purposes because of the non-optimal recovery rate.

The state of the art in white-box analysis is the work by Bunker et al. [8] and [17], who built a complete custom setup to interact with flash memory chips using an FPGA and several custom wing boards to enhance its compatibility. Although their goal is to enable easy development and prototyping of FTL algorithms, compression, cryptography and sanitization protocols, the same setup can be theoretically used to re-implement part of the FTL functionalities to ease white-box acquisition of SSDs. However, the internals of a controller are usually undocumented; therefore, it may be very difficult to reconstruct files directly from the acquired data, and traditional file carvers are likely to fail. Although Billard and Hauri [6] showed how is possible to analyze a raw flash dump and reconstruct files without prior knowledge of the disposition of blocks performed by the FTL, their technique works only with small capacity chips (in the order of hundreds of megabytes).

Luck and Stokes [14] concentrated on FAT structure and demonstrated how to rebuild audio and video files from dumped NAND memories; their work suffers from the very same shortcoming: It is tailored for small amounts of data (e.g., cellphone memories). Also, data reconstruction is made even more difficult by SSD controllers because they often make use of data parallelism over the flash memory chips on the board.

Differently from white-box techniques, our proposed methodology is extremely convenient, practical and, most importantly, guarantees that the SSD is never damaged.

7.2 Black-box Forensics Analysis

Similarly to Bell and Boddington [5], King and Vidas [13] performed experiments on 16 different SSDs: They simulated real usage scenarios and tested the block-level recoverability. Each scenario was replicated under three OSs (Windows XP, Windows 7 and Ubuntu Linux 9.04). Their conclusion is that different combinations of usage, OS and file size influence the forensic recoverability of the SSD. Although this is by far the most exhaustive test on SSDs, the authors focus solely on data deletion as effect of TRIM and garbage collection, without generalizing their findings. What is missing is an in-depth study of the correlation between the environment conditions (e.g., OS, filesystem, file size), the internal state of a disk (e.g., amount of free space, wear) and the corresponding behavior of the SSD. Our work goes beyond [13] because we designed and evaluated a comprehensive, test-driven methodology to fully understand the reasons behind each specific behavior.

Antonellis [4] analyzed the behavior of one of the first SSDs with respect to file deletion. He wrote some typical files such as documents and images on an NTFS-formatted SSD, and then erased them to see how much data was recoverable afterward. Surprisingly,

none of the files was recoverable via carving. These experiments, however, were narrowed to a single scenario and did not take into account all the possible factors that our methodology accounts for. For instance, as detailed in §3.2 and §4, thanks to our methodology we can explain why caching is the most reasonable explanation of Antonellis [4]'s odd results. Last, [4] focused on TRIM, whereas our methodology considers the features implemented in most of the SSDs currently on the market.

8. CONCLUSIONS

In order to overcome the intrinsic limitations of SSDs, onboard controllers adopt a number of advanced strategies, preemptively erasing blocks of deleted files (possibly even if not solicited by the OS) and even performing compression or encryption on the data. Each vendor implements a different controller, and therefore each SSD acts differently. Consequently, SSDs cannot be treated as standard HDDs when performing a forensic analysis: Standard tools and well-known techniques are based on the assumption that the hard drive does not modify or move data in any way, that every block can be read if it was not previously wiped, and that reading a block will yield the data physically contained by that block.

We proposed a complete testing methodology and applied it to three drives of leading vendors. For each test we interpreted the results to provide the forensic analyst a practical way to fine tune their approaches to the acquisition of SSD drives, which can be very similar to acquisition of HDDs, or completely different. Indeed, we showed that the combination of controller, OS, filesystem and even disk usage can deeply influence the amount of information that can be retrieved from a disk using forensic procedures. We advanced the state of the art by showing further, previously-unknown findings, such as the one described in §4.1: even though all the drives expose the TRIM functionality to the OS, our methodology allowed us to investigate the peculiarities of each implementation (e.g., a drive selectively wiping blocks only in specific portions of it). We also proposed a test aimed at verifying the implementation of a compression feature on the SSD. We showed that our test can indeed devise whether the drive uses compression or not; this is useful to determine the feasibility of a deeper white-box approach. We also investigated the controversial topic of garbage collection to see under what conditions it triggers.

Each of the proposed experiments is tailored to inspect how each functionality works and how "aggressive" it is. They allow the analyst to know, for example, if the SSD has a disruptive TRIM implementation or if GC does not work under particular conditions. All these information can be of great help when performing a forensic acquisition on such drives, since they can change the expectations or even suggest some particular techniques to adopt in order not to allow the controller to delete possible useful data. Also, the result can help the forensic expert to estimate the success of costly procedures such as a memory white-box analysis.

Besides addressing the aforementioned limitations, future work can focus on running our methodology on a wide range of OSs, SSD brands and models, in order to create a reference catalog useful for forensic investigations.

REFERENCES

[1] Wear leveling in micron NAND flash memory. Technical Report TN-29-61, Micron Technology Inc., 2008. URL http://www.micron.com/~/media/Documents/Products/Technical%20Note/NAND%20Flash/tn2961_wear_leveling_in_nand.pdf.

[2] Wear-leveling techniques in NAND flash devices. Technical Report TN-29-42, Micron Technology Inc., 2008. URL http://www.micron.com/~/media/Documents/Products/Technical%20Note/NAND%20Flash/tn2942_nand_wear_leveling.pdf.

[3] Microsoft Co.: Support and Q&A for Solid–State Drives. MSDN Blog http://blogs.msdn.com/b/e7/archive/2009/05/05/support-and-q-a-for-solid-state-drives-and.aspx, 2009.

[4] Christopher J. Antonellis. Solid state disks and computer forensics. *ISSA Journal*, pages 36–38, 2008.

[5] Graeme B. Bell and Richard Boddington. Solid state drives: The beginning of the end for current practice in digital forensic recovery? *Journal of Digital Forensics, Security and Law*, 5 (3), December 2010.

[6] David Billard and Rolf Hauri. Making sense of unstructured flash-memory dumps. In *SAC '10*, pages 1579–1583, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-639-7.

[7] Marcel Breeuwsma, Martien De Jongh, Coert Klaver, Ronald Van Der Knijff, and Mark Roeloffs. Forensic data recovery from flash memory. *Small Scale Digital Device Forensics Journal*, 1:1–17, 2007.

[8] Trevor Bunker, Michael Wei, and Steven Swanson. Ming II: A flexible platform for NAND flash-based research. Technical Report CS2012-0978, UCSD CSE, 2012.

[9] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo. Improving flash wear-leveling by proactively moving static data. *Computers, IEEE Transactions on*, 59(1):53 –65, jan. 2010. ISSN 0018-9340.

[10] Jim Gray and Bob Fitzgerald. Flash disk opportunity for server applications. *Queue*, 6(4):18–23, July 2008. ISSN 1542-7730.

[11] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman Pletka. Write amplification analysis in flash-based solid state drives. In *SYSTOR '09*, pages 10:1–10:9, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-623-6.

[12] Intel. AP-684: Understanding the flash translation layer (FTL) specification. Intel Application Note. http://www.jbosn.com/download_documents/FTL_INTEL.pdf, 1998.

[13] Christopher King and Timothy Vidas. Empirical analysis of solid state disk data retention when used with contemporary operating systems. volume 8, pages S111–S117, Amsterdam, The Netherlands, The Netherlands, August 2011. Elsevier Science Publishers B. V.

[14] James Luck and Mark Stokes. An integrated approach to recovering deleted files from nand flash data. *Small Scale Digital Device Forensics Journal*, 2(1):1941–6164, 2008.

[15] Sergei P. Skorobogatov. Data remanence in flash memory devices. In *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th Intl. Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proc.*, volume 3659 of *Lecture Notes in Computer Science*, pages 339–353. Springer, 2005.

[16] Robert Templeman and Apu Kapadia. Gangrene: exploring the mortality of flash memory. In *HotSec'12*, pages 1–1, Berkeley, CA, USA, 2012. USENIX Association.

[17] Michael Wei, Laura M. Grupp, Frederick E. Spada, and Steven Swanson. Reliably erasing data from flash-based solid state drives. In *FAST'11*, pages 8–8, Berkeley, CA, USA, 2011. USENIX Association. ISBN 978-1-931971-82-9.