Testing BIOS Interrupt 0x13 Based Software Write Blockers

James R. Lyle and Paul E. Black

Keywords: Acquisition of digital data, software testing, testing forensic tools, write blocking.

Contact: Paul E. Black, paul.black@nist.gov, 1 301 975 4794
100 Bureau Drive Stop 8970
Gaithersburg, MD, USA 20899-8970

James R. Lyle, JLYLE@NIST.GOV, 1 301 975-3270
100 Bureau Drive Stop 8970
Gaithersburg, MD, USA 20899-8970

1 Introduction

This paper reports observations and experience in the Computer Forensics Tool Testing (CFTT) project at the U.S. National Institute of Standards and Technology (NIST) while developing methodologies for testing software write block (SWB) tools. CFTT is a joint project of the National Institute of Justice (NIJ), the research and development organization of the U.S. Department of Justice; NIST Office of Law Enforcement Standards (OLES) and Information Technology Laboratory (ITL); and is supported by other U.S. government organizations. The goal of the CFTT project is to establish a methodology for testing computer forensics tools.

The goal of a write blocker is to allow an investigator to read to all digital data on a storage device while preventing any changes to the storage device. A write blocker is typically used either to protect a hard drive during preview of the drive contents prior to acquiring the contents or to protect the drive during the acquisition process. The basic strategy to implement a write blocker is to place either a hardware or software filter somewhere between application programs and the storage device that is to be protected. The filter intercepts I/O commands sent to the hard drive and only allows commands that make no changes to the device.

1.1 Different Kinds of Write Blockers

Such a filter can be implemented either in software or in hardware. A software write block (SWB) program may have several advantages over a hardware write block device: easier to design and implement, easier to install (opening the computer case is not required), and easier to test. However, a SWB tool also has a number of disadvantages: unless the SWB runs at the very lowest level and the operating system is secure, there are usually ways for other programs to subvert a software write blocker. Programs may use alternate or lower level access, even writing directly to hardware ports, to avoid the SWB altogether. Programs may also intentionally or accidentally disable SWB protection. A software write blocker is easier to use incorrectly and slows the acquisition process.

This paper only discusses testing software write blockers based on interrupt 0x13 BIOS requests.

1.2 A Brief Review of Interrupt 0x13 Disk Access

Disk drives are accessed by sending commands from the computer to a drive controller device, which actually runs the disk drive. Since the low level programming required to use the drive controller directly is difficult, tedious, and not portable, operating systems usually provide other access interfaces. For example, programs running in the MS-DOS®[1] environment can access the

[1] **Certain trade names and company products are mentioned in the text or identified. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are necessarily the best available for the purpose.**
MS-DOS is a registered trademark of Microsoft Corporation.

disk drive via the drive controller, the MS-DOS service interface (interrupt 0x21), or the BIOS service interface (interrupt 0x13).

To use BIOS service an application program issues an interrupt 0x13 command. The interrupt transfers control to the BIOS interrupt 0x13 routine. The BIOS routine issues commands directly to the disk drive controller. The device does the requested operation and the controller returns the result to the BIOS which then returns it to the application program.

A SWB tool changes the normal operation of the interrupt 0x13 interface. When first executed, the SWB tool saves the address of the old interrupt 0x13 routine and installs a new interrupt 0x13 routine. Subsequently interrupt 0x13 commands issued by application programs are handled by the SWB tool. If the command should be allowed, the command is passed to the old handler. If the command should be blocked, the SWB tool returns to the application program without passing any command to the BIOS interrupt 0x13 routine. Depending on SWB tool configuration, either **success** or **error** is returned for the command status code.

1.3 The Specification of a Software Write Block Tool

Precise, objective testing begins with a clear specification. Using focus groups, NIST developed a specification of a general interrupt 0x13 software write block tool [1]. Although simple enough in the abstract, the detailed specification has subtleties. For instance, the philosophy of SWB tools has evolved over the years. The original design of *only block known writes* has given way to *only allow known reads* design. The latter is safer, for instance, when a new write command is added to the BIOS command set.

As another example, the specification classifies all commands in one of six categories: read, control, information, write, configure and miscellaneous. Read, control, and information commands do not change the stored data, by definition, and hence must always be allowed. Commands in the write category must be blocked. However, configure commands are a problem: they can change the way a drive appears to the host. If a configuration command changes the apparent size of a drive, part of the drive might become inaccessible to read commands. However, a configuration command might be needed to make an inaccessible portion of a drive readable. The miscellaneous category has similar problems. The SWB specification requires all configuration and miscellaneous commands to be blocked. Even after developing a specification, deciding the number, type, and constitution of test cases was challenging.

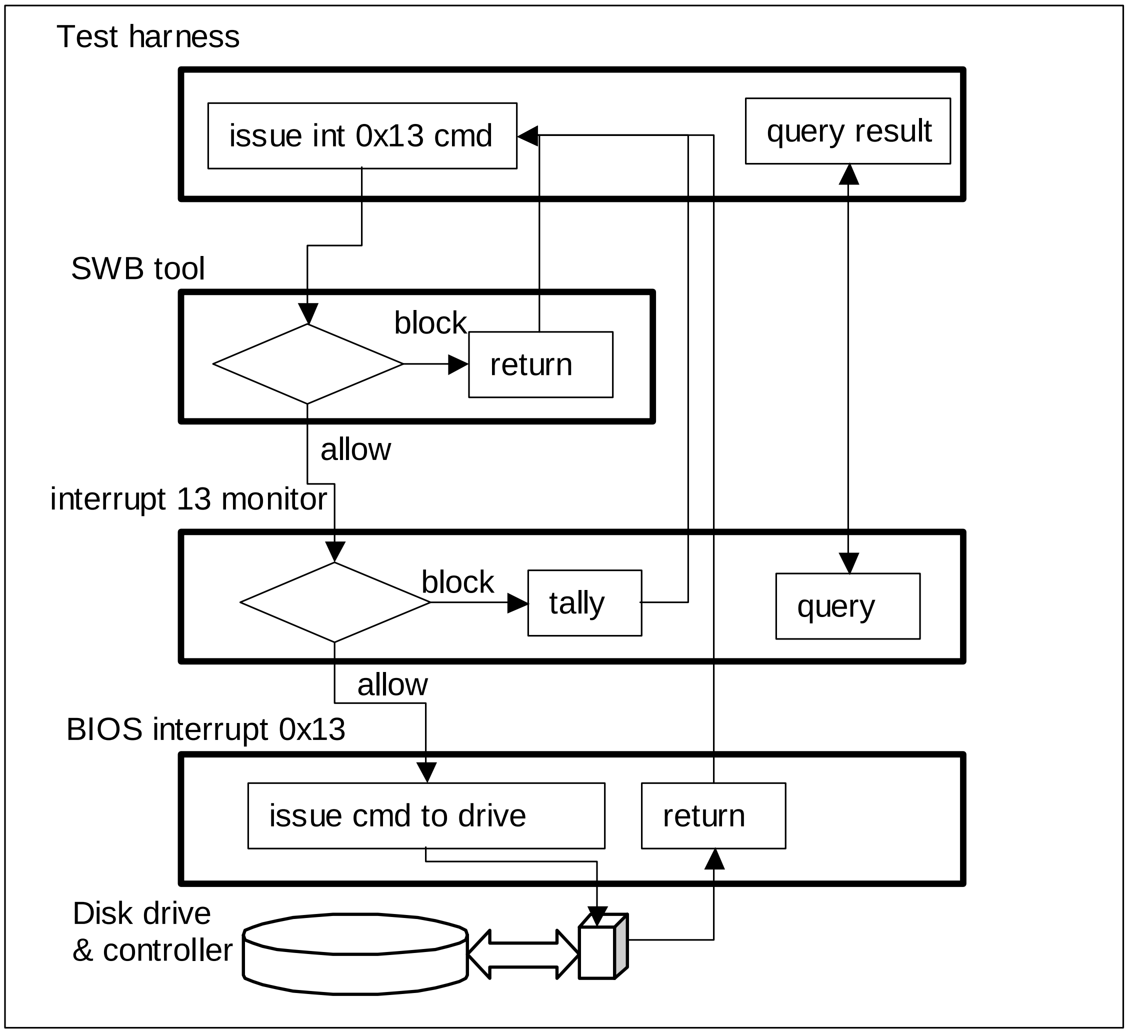
2 Methods

This section describes the method used to test software write block tools, which uses special test programs and scripts, and the method to validate the test programs.

2.1 A Method to Test Interrupt 0x13 SWB Tools

The core of the method is a two-piece wrapper around the SWB tool, illustrated in Figure 1. The test harness sends the desired sets of BIOS interrupt 0x13 commands to the SWB tool. The interrupt 0x13 monitor intercepts any commands allowed by the SWB tool, preventing any actual change to disks on the test machine, and counts the number of commands allowed. The harness queries the monitor to infer which commands the SWB tool allowed and which it blocked. In start-up mode, the monitor allows commands to pass so the SWB tool can get information about the disks.

Figure 1 Operation of the Test Harness and Interrupt Monitor While Testing an SWB Tool[2]



In more detail, the normal interrupt 0x13 BIOS routine is first overlaid with an interrupt 13 monitor, called tally13. Tally13 also overlays the interrupt 0x17 BIOS routine, normally used for printing, to respond to queries. Tally13 operates in one of two modes: passive and active. In

[2] The test harness issues commands to the SWB tool under test, which blocks or allows commands. Commands which are allowed to reach the interrupt 13 monitor are tallied, if the monitor is active. If the monitor is in passive mode, commands are allowed to reach the normal BIOS interrupt 0x13 handler. The test harness queries the monitor for the number of commands received.

passive mode all commands are passed to the BIOS interrupt which passes them to the disk drive. Tally13 starts in passive mode and must be switched to active mode. After tally13 has started, the SWB tool under test is started. After the SWB tool has started, the test harness is executed. All the components of the testing setup are now ready, as diagrammed in Figure 1.

The test harness, called test-hdl, first switches tally13 to active mode via the interrupt 0x17 query interface. In active mode tally13 blocks all commands and counts the number of times each I/O commands is received for each disk drive. Now any command can be safely issued to a SWB tool whether it blocks or allows it, and we do need to test that in some modes the SWB tool allows commands.

The test harness issues every command in the category indicated when it started. For example, a test of the read command category issues every read command defined: 0x02 (read), 0x10 (read long) and 0x42 (extended read). As each command is issued, the SWB tool receives the command and either blocks (returns with no action) or allows the command (passes it on to the interrupt 0x13 monitor, tally13). If the command reaches tally13, a counter for each received command and disk is incremented and tally13 returns. After control returns to the test harness, the test harness uses tally13's interrupt 0x17 query interface to get the number of times the issued command was received. If the count is zero, the SWB tool blocked the command. A non-zero count indicates that the command was not blocked, assuming nothing else sent commands.

One test case consists of

1. rebooting the test computer,
2. starting tally13,
3. starting the SWB tool with appropriate command line arguments,
4. starting test-hdl with command line arguments for logging information and the command categories to be run, and
5. capturing the operator's observations with a program called sig-log and saving various log and output capture files.

The complete test plan consists of 40 test cases. Each case is designed to provide information to support or refute assertions and to exercise the SWB tool in many different configurations.

2.2 Additional Programs and Scripts to Run the Tests

The testing programs take identifying information, such as the operator name, computer name, and test case, as argument on the command line and write this and other information to log files. To eliminate the possibility of some human errors, a batch file script runs the entire test case to

1. invoke tally13, the SWB tool, test-hdl, and sig-log in the right order,
2. capture their outputs in log files,
3. consistently pass the identifying information, such as the test case, to the programs, and
4. save all the log files of one test case in a single directory, which is named for the test case.

Because of fundamental differences between the test cases that cannot be handled with a single script, there are several variants. For instance, to exercise boot-time use of the SWB tool, a special autoexec.bat file is run. Because of differences in invocation and reporting, slight variants were necessary between tools and some versions of some tools.

2.3 *Validating the Test Tools*

The testing tools and the SWB tool interact by means not easily observed. How much confidence should we have that tally13 and test-hdl behave properly, that is, that they are testing the SWB tool according to the test plan? To address this, the testing tools themselves were validated independently with an emphasis on watching for errors that may allow invalid results. This document uses the term “validate”, instead of test, for several reasons. First, it may be confusing to talk about, for instance, test case test cases, that is, cases to test other test cases. Also, one of the methods is manual code inspection or review, not executable tests. Most importantly, standard documents say that institutions “shall validate non-standard methods” [2] Sect. 5.4.5.2.

The entire method, the plan and the tools together, could be validated. That is, the plan and the tools would be checked against a general set of requirements for SWB test tools. This comprehensive approach would take a lot of time and effort. Instead each tool has its own, separate, explicit requirements, against which it is validated.

One possible approach is to essentially repeat what has been done, by creating independent versions of the tools and plan, test the same SWB tools again, and compare the results. This would lack crucial independence: the examiner would be writing similar tools for similar purposes and running similar tests on similar machines. We would expect the examiner to be prone to make “equivalent logical errors” or different logical errors with “statistically correlated failures” [3].

Instead the approach includes several different methods to validate the SWB test tools. Code is reviewed or inspected for possible anomalous behavior [4] and to determine if it meets its requirements. Compiled versions of the programs are executed in very simple, controlled tests to independently check behavior. Information from code reviews and information from test runs are combined to support assertions.

These methods are likely to find simple errors, but will they find complex, subtle, or non-localized errors? Researchers have found support [5] [6] [7] for the “fault coupling hypothesis”: tests for simple faults are likely to find complex faults, too. Thus there is support that checking for simple errors suffices.

The tools are divided into three groups so they can be validated in phases. As pointed out above, duplicating work already done to develop SWB test tools increases the cost of tests and doesn’t increase independence much. However the nature of interrupt 0x13 service handling is such that validating the SWB test tools requires much the same functionality as the SWB test tools themselves!

For instance, clearly the examiner needs to check that the test harness sends the intended interrupt 0x13 commands. Other than inspecting the code, one simple strategy would be to run the driver then examine the disk drive for evidence of the commands. This has the limitation that only effects of certain command parameters are measured: some status commands are defined to be handled by the BIOS without sending anything to the disk. A second limitation is that some commands should only be executed in a factory setting. Some commands have subtle and undocumented results. The proper parameter values are propriety, and the user is cautioned that improper parameters may render a disk drive unusable. What is a good way to test that the driver sends such potentially destructive commands without the risk of damaging disk drives? Intercept interrupt 0x13 commands and report how many and which kind were sent. But this is essentially the function of tally13.

3 Results

In this section we first report the results of validating the testing tools. Then we report some of our experience testing seven software write block tools and a selection of the results from the testing.

3.1 Results of Validating the Support Tools

To begin the actual validation of the testing tools, we reviewed the source code for tally13, test-hdl, sig-log, and t-off and tested the compiled versions of these tools. We found some coding and execution anomalies and made some observations [8].

One anomaly found during code review is that if used with more than five disk drives, tally13 overwrites memory. The program allocates enough memory for counts for five disk drives and does not check that commands are only for one of the first five. Since we never used more than five disk drives at once, this was not a problem in our tests. Even a configuration of six or more disk drives is unlikely to lead to invalid SWB tool evaluation. If the overwrites are not in critical memory areas, for instance, the start-up message buffer, operation may be correct. Otherwise low-level memory overwrites are likely to crash the system or cause blatantly incorrect results. Since tally13 is only meant for expert use and the results are subject to human inspection, leaving out code to detect commands for additional drives is justified.

Another anomaly found during code review and confirmed by execution is that if interrupt 0x13 commands are handled in extremely unorthodox ways, test-hdl might not accurately characterize the behavior in terms of blocking or allowing commands. Test-hdl checks that command counts are zero, sends all commands ordered, then checks the command counts again. If the count for a command is zero, test-hdl classifies the command as blocked. If the count is not zero, it is classified as allowed. If the SWB tool receives a command, but passes on a different command or commands are sent from other sources, test-hdl's classifications will not be correct. Some scenarios could be eliminated if test-hdl checked counts immediately before and after sending each command. For SWB tools that always either block or allow commands depending on the tool's mode, test-hdl characterizes the behavior well.

We observed that some of the requirements and assertions for the testing support tools were poor. The support tools did not strictly conform to some assertions, but this nonconformance had no effect on the utility of support tools in testing SWB tools. For instance, STV-RM-11 requires that test-hdl reports if any command counts are not zero and if so, *which* command is not zero [9]. Since it does not matter which command is not zero, the requirement is better stated as: “If the number of each command received for each disk drive is not zero, test-hdl logs the ~~command and~~ disk drive.”

3.2 Experience Testing SWB Programs with the Support Tools

Because of differences in invocation and reporting, slight changes were necessary to the test procedures and scripts between tools and some versions. More importantly, no two versions behaved in exactly the same way: each tool or version blocked or allowed a slightly different set of commands or behaved differently in some operational aspect. All the tools blocked the same core set of write commands, but there were treatments varied for the configuration and miscellaneous categories.

As mentioned in Section 2.2, testing uses scripts to invoke the tools in the right order, to consistently pass identifying information, and to capture and save the output. Because the SWB tools tested use different OS mechanism to produce information, we had to change the scripts slightly. We could redirect the output of one tool to a file and capture all its output. A variant script invoked the other SWB tool more than once: the first time to start it and again to capture information it reports.

In spite of having additional scripts to automate much of the testing and program invocation, testers still had to rerun tests. In one instance the tester misunderstood the parameters needed for a test run. In another instance, the tests were run designating a fifth drive, but the tool aborted as that designation is not allowed. Since most of the output was captured, the tester did not notice the invalid runs until they were reviewed much later! Some tests could not be considered reliable since the hash signature of a disk’s content after the tests did not match the hash before the test. (An operator error in booting from a disk between tests is the most likely source of the change.) In all cases we reran tests when necessary to get reliable tests run correctly, which were the basis of the final reports.

We also ran some additional tests to investigate concerns that arose during the testing. These tests helped us determine, for example, if an unexpected result came from an operator error in running a test, an incorrect machine configuration, the behavior of the SWB tool, etc.

3.3 Results of Testing SWB Programs

Seven software write block tools have been tested: four versions of one tool, HDL, and three versions of a second, PDBLOCK.

Although it blocked all commands that its documentation said it blocked, HDL V0.4 did not block some commands that could change the content or accessibility of a disk drive [10]. These commands include a newer command and commands that are rarely, if ever, used by disk imaging programs. HDL V0.5 blocked more of these commands, but not all [11]. The next version tested, HDL V0.7, blocked all but two configuration commands [12]. It also blocked two commands that could have been allowed. The latest version tested, HDL V0.8, blocked all commands that could change the disk content or accessibility, but also blocked two commands that could have been allowed [13].

PDBLOCK V1.02 (PDB_LITE) blocked all write commands, but did not block five commands in the configuration category and two commands in the miscellaneous category [14]. It also did not report accessible drives, as required. The full version, PDBLOCK V2.00, did not report accessible drives, either [15]. PDBLOCK V2.00 blocked all write commands. There were five configuration commands and three miscellaneous commands it did not block. The third miscellaneous command, 0xED, is not used by any disk drive we know. PDBLOCK uses that command to change the mode of a running process. The next version we tested, PDBLOCK V2.10, blocks and allows the same commands as version 2.00, but it does report accessible drives [16]. We observe that for some command line specification of protected subsets, version 2.10 did not give specified protection status to a fifth disk drive. This is not an anomaly since one line in the documentation may imply that PDBLOCK V2.10 should not be used with more than four disk drives.

4 Discussion

Seven programs have been evaluated with the behavior of software write block (SWB) tools specified. The evaluations bring out some of the limitations of each tool. More importantly, the evaluations support that they are useful and reliable tools.

Similar projects to specify and evaluate other computer forensics capabilities, such as disk imaging, hardware write blockers, deleted file recovery, and string searching, are in various stages. These help to more firmly ground the work that is done.

The specifications, although carefully written, are not completely unambiguous and sometimes do not convey the precise meaning needed. Efforts, like that of Andreas Enbacka (personal communication), to use more formal specifications should reduce ambiguity, increase the precision of definitions, and minimize gaps. When used with commentary, such specifications may be able to quickly orient a human reading them, but also more exactly convey details.

Even when used successfully, software tools can benefit from third-party evaluation. Such evaluations confirm the capability of tools, provide a consistent framework for comparing tools, and may help tool developers determine enhancements that are most helpful to the community.

• References

[1] Software Write Block Tool Specification & Test Plan, Version 3.0, September 1, 2003. http://www.cftt.nist.gov/documents/SWB-STP-V3_1a.pdf accessed 24 February 2004.

[2] NIST Handbook 150, 2001 ed. “National Voluntary Laboratory Accreditation Program; Procedures and General Requirements.”

[3] Susan S. Brilliant, John C. Knight and Nancy G. Leveson, “Analysis of Faults in an *N*-Version Software Experiment”, IEEE Transactions on Software Engineering, 16(2):363-377, February 1990.

[4] Jorge Rady de Almeida Jr., João Batista Camargo Jr., Bruno Abrantes Basseto, and Sérgio Miranda Paz, “Best Practices in Code Inspection for Safety-Critical Software”, IEEE Software, 20(3):56-63, May/June 2003.

[5] Richard A. De Millo, Richard J. Lipton and Frederick G. Sayward, “Hints on Test Data Selection: Help for the Practicing Programmer”, IEEE Computer, 11(4):34-41, April 1978.

[6] A. Jefferson Offutt, "Investigations of the Software Testing Coupling Effect", ACM Trans. on Software Engineering Methodology, 1(1):3-18, January 1992.

[7] D. Richard Kuhn, Dolores R. Wallace, and Albert M. Gallo, Jr., “Software Fault Interactions and Implications for Software Testing”, IEEE Transactions on Software Engineering, 30(6):418-421, June 2004.

[8] Paul E. Black, “Software Write Block Testing Support Tools Validation: Test and Code Review Report”, NIST IR 7207B, 2005.

[9] Paul E. Black, “Software Write Block Testing Support Tools Validation: Test Plan, Test Design Specification, and Test Case Specification”, NIST IR 7207A, 2005.

[10] “Test Results for Software Write Block Tools: RCMP HDL 0.4”, National Institute of Justice, August 2004, http://www.ncjrs.org/pdffiles1/nij/206231.pdf accessed 19 November 2004.

[11] “Test Results for Software Write Block Tools: RCMP HDL 0.5”, National Institute of Justice, August 2004, http://www.ncjrs.org/pdffiles1/nij/206232.pdf accessed 19 November 2004.

[12] “Test Results for Software Write Block Tools: RCMP HDL 0.7”, National Institute of Justice, February 2004, http://www.ncjrs.org/pdffiles1/nij/206233.pdf accessed 19 November 2004.

[13] “Test Results for Software Write Block Tools: RCMP HDL 0.8”, National Institute of Justice, August 2004, http://www.ncjrs.org/pdffiles1/nij/203196.pdf accessed 19 November 2004.

[14] “Test Results for Software Write Block Tools: PDBLOCK Version 1.02 (PDB_LITE)”, to be published.

[15] “Test Results for Software Write Block Tools: PDBLOCK Version 2.00”, to be published.

[16] “Test Results for Software Write Block Tools: PDBLOCK Version 2.10”, to be published.