SWBT 1.0: Software Write Block Testing Tools

Requirements, Design Notes and User Manual

Version 1.0
October 2003

NIST
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

Abstract[†]

This document describes Release 1.0 of SWBT a software package developed to aid the testing of software write block tools typically used in forensic investigations. The software can be used in the DOS environment to test an interrupt 0x13 based write block tool, measure the results and aid in documenting test runs. The package includes programs that monitor the interrupt 0x13 BIOS disk interface and send selected commands to a software write block tool under test. The software is written in either Borland C++ 4.5 or Borland Assembler.

The intended audience for this document should be familiar with the DOS operating system, computer operation, computer hardware components such as hard drives, hard drive interfaces (e.g., IDE or SCSI) and computer forensics. A working knowledge of C and Assembly programming is not necessary for understanding but would be helpful.

[†] **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.**

Contents

Table of Figures

List of Tables

1 Introduction

The Computer Forensics Tool Testing (CFTT) program is a joint project of the National Institute of Justice (NIJ), the research and development organization of the U.S. Department of Justice; NIST's Office of Law Enforcement Standards (OLES) and Information Technology Laboratory (ITL); and is supported by other organizations, including the Federal Bureau of Investigation, the Department of Defense Cyber Crime Center, and the Department of Homeland Security's Bureau of Immigration and Customs Enforcement and U.S. Secret Service. The objective of the CFTT project is to provide measurable assurance to practitioners, researchers, and other applicable users that the tools used in computer forensics investigations provide accurate results. Accomplishing this requires the development of specifications and test methods for computer forensics tools and subsequent testing of specific tools against those specifications.

This document describes Release 1.0 of the SWBT tools package (Software Write Block Testing), a test harness for testing interrupt 0x13 based software write block (SWB) tools. A SWB tool is a software tool that is used to protect a hard drive from modification, usually during a forensic examination of the content of the hard drive. The user of the SWB tool may wish to examine the hard drive content or the user may wish to use the SWB tool in conjunction with a hard drive imaging tool to create a forensic image of the hard drive for later analysis. The SWBT package includes a program (tally13.com) that monitors interrupt 0x13 to report commands allowed or blocked, a program to deactivate tally13.com (t-off.exe), a program to record operator observations of audio or visual signals from the tool under test (sig-log.exe) and a program (test-hdl.exe) to send specific groups of commands to the software write block tool under test. The test-hdl.exe, sig-log.exe and t-off.exe programs are written in Borland C++ 4.5 and tally13.com is written in Borland Assembler. The software can be used in the DOS environment to test programs such as RCMP HDL. A set of test cases for software write block tools is described in *Software Write Block Tool Specification & Test Plan Version 3.0* (see http://www.cftt.nist.gov/).

1.1 How to read this document

The intended audience for this document should be familiar with the DOS operating system, computer operation, computer hardware components such as hard drives, hard drive interfaces (e.g., IDE or SCSI) and computer forensics. A working knowledge of C and Assembly programming is not necessary for understanding but would be helpful. We expect the primary objectives of the reader to be some combination of the following:

1. The reader wants a general overview of the SWBT software.
2. The reader wants to use the SWBT software as is.
3. The reader wants to use SWBT after enhancement or modification.
4. The reader wants to test the suitability of SWBT as a test harness.

The remainder of this section describes the capabilities required to test software write block tools and lists the components of SWBT. The goal is to identify capabilities

required for testing software write block tools and allocate the required capabilities to the programs of the SWBT package.

Section 2 documents the requirements of each program and gives a concise description of what each program is supposed to do.

The program designs are discussed in **Section 3**. This section should be read in conjunction with the program listings. The objective of this section is to give the reader a high level view of the program design to help understand the source code.

Section 4 is the user manual describing command line parameters and program outputs.

Not all the document is of interest to every reader. The goals of the reader determine the relevant sections of the document. A bullet (**●**) in Table 1-1 indicates the relevant sections given the goals of the reader.

Table 1-1 Reader's Guide

Reader's Goal	Section			
	1	2	3	4
General overview of SWBT	●			
Use SWBT to test a software write block tool	●			●
Enhance or modify SWBT	●	●	●	●
Verify or test the operation of SWBT as a test harness	●	●	●	●

1.2 Capabilities Required to Support Testing Software Write Block Tools

This section establishes the capabilities required to setup and measure each test. This is accomplished by first determining what must be measured to confirm compliance with the tool requirements and second what conditions must be created to allow an accurate measurement of compliance.

The basic function of a software write block tool is to not allow a protected drive to be modified while still allowing any access that does not modify the drive. The critical requirements are the following:

- The tool shall not allow a protected drive to be changed.
- The tool shall not prevent obtaining any information from or about any drive.
- The tool shall not prevent any operations to a drive that is not protected.

The next subsection presents an overview of how hard drives are physically attached to a computer and then accessed by programs running on the computer. The following subsection defines terminology related to software write block tools.

1.2.1 Hard Drive Attachment and Access

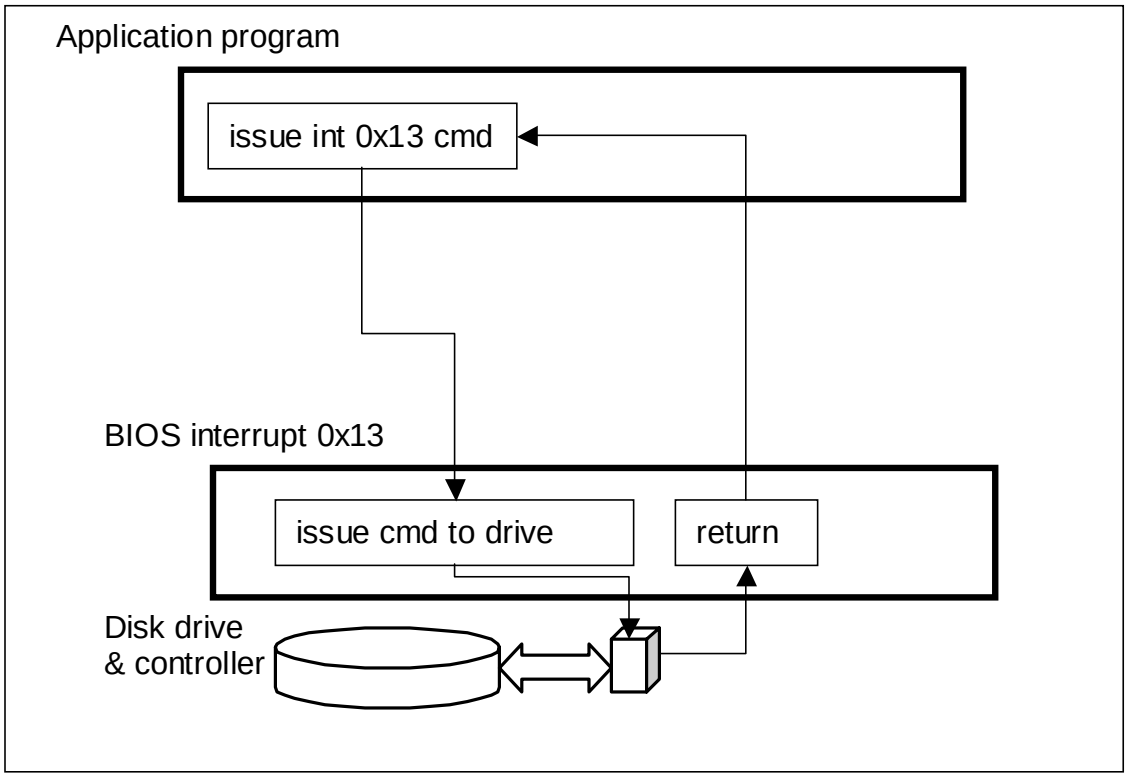
Before a hard drive can be used it must be physically attached to a computer. A hard drive is attached to a computer by one of several available physical interfaces. A drive is

usually connected by a cable to a drive controller located either on the mother board or on a separate adapter card. The most common physical interface for communicating with the hard drive through the drive controller is the ATA (AT Attachment) or IDE (Integrated Drive Electronics) interface. This includes variants such as EIDE (Enhanced IDE) or ATA-2, ATA-3, etc. Some other physical interfaces include, but are not limited to SCSI (Small Computer System Interface), IEEE 1394 (also known as FireWire or i-Link), and USB (Universal Serial Bus).

All access to a drive is accomplished by commands sent from a computer to a drive through the drive controller. However, since the low level programming required for direct access through the drive controller is difficult and tedious, each operating system usually provides other access interfaces. For example, programs running in the DOS environment can, in addition to direct access via the drive controller, use two other interfaces: DOS service interface (interrupt 0x21) or BIOS service interface (interrupt 0x13). The DOS service operates at the logical level of files and records while the BIOS service operates at the physical drive sector level. More complex operating systems, for example Windows XP or a UNIX variant (e.g., Linux), may disallow any low level interface (through the BIOS or the controller) and only allow user programs access to a hard drive through a device driver, a component of the operating system that manages all access to a device.

Using the interrupt 0x13 interface for hard drive access is illustrated in Figure 1-1. An application program issues an interrupt 0x13 command. The interrupt transfers control to the interrupt 0x13 routine in the BIOS. The BIOS routine issues commands, ATA or SCSI as appropriate, directly to the hard drive controller. The device does the requested operation and returns the result to the BIOS and then to the application program.

Figure 1-1 Drive Access Through the 0x13 BIOS Interface



1.2.2 Technical Terminology

A hard drive software write block tool replaces or monitors a hard drive *access interface* on a general purpose host computer with hard drives attached by a physical interface. A hard drive access interface is defined as a method used by a program to access a hard drive. For a program to access a drive, the program issues a high level command to the access interface that is translated by the access interface into the corresponding low level command that is sent to the drive through the drive controller. For each command issued, the access interface indicates command results (e.g., command completion, error status) by a *return value*. A hard drive software write block tool operates by monitoring drive I/O commands sent from the PC through a given access interface. Any commands that could modify a hard drive are intercepted (i.e., blocked) and not passed on to the hard drive controller. The most common access interfaces currently found are as follows: hard drive device driver, interrupt 0x13 BIOS (Basic Input Output Services), ATA (AT Attachment) direct controller, ASPI (Advanced SCSI Programming Interface), USB (Universal Serial Bus) and IEEE 1394 (also known as Firewire). Each interface has its own command set and access protocol. The command set for a given interface can be partitioned into the following categories:

- Write: commands that transfer data from the computer memory to the drive.
- Configuration: commands that change how the drive is presented to the host computer. These commands often destroy data on the drive or make data inaccessible.

- Read: commands that transfer data from the drive to the computer memory.
- Control: commands that request the drive to do a nondestructive operation, for example: reset or seek.
- Information: commands that return information about the drive.
- Miscellaneous: commands that do not fit into the other categories**.**

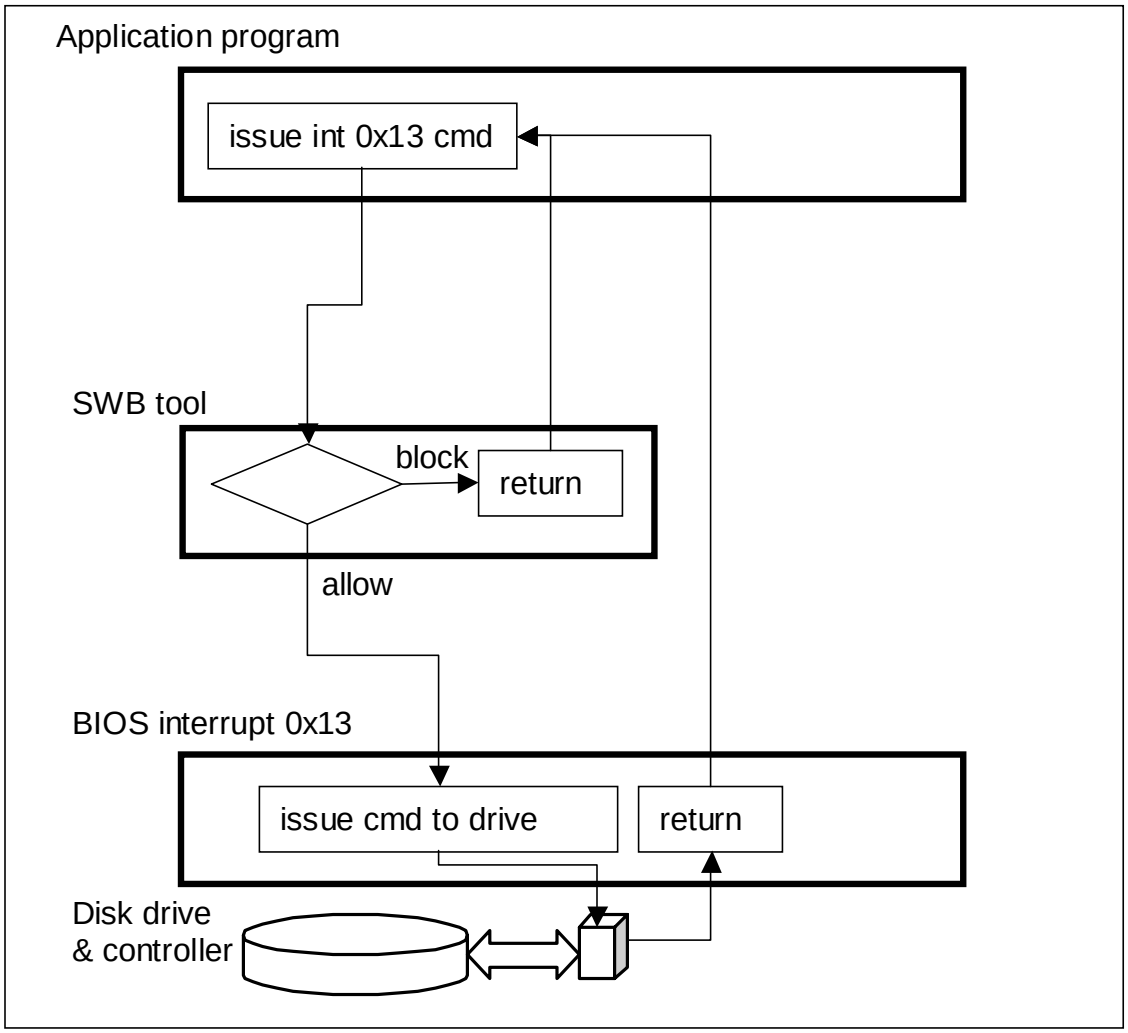
The following terms are defined for convenience in specifying the tool requirements.

- Covered interface: a drive access interface that is controlled by the SWB tool under test.
- Covered drive: a drive attached to a covered interface.
- Protected drive: a drive designated for protection from modification when accessed by a covered interface.
- Unprotected drive: a drive that is not protected from modification through a specified access interface.
- Blocked command to a drive: a command issued by an application program that is intercepted by a SWB tool such that neither the issued command nor some other command is sent to the drive. A command that is not blocked is not altered in any way.

Use of a SWB tool changes the normal operation of the interrupt 0x13 interface. Figure 1-2 illustrates SWB tool operation.

1. The SWB tool is executed. The SWB tool saves the current interrupt 0x13 routine entry address and installs a new interrupt 0x13 routine.
2. The application program initiates a drive I/O operation by invoking interrupt 0x13. The replacement routine installed by the SWB tool intercepts the command.
3. The SWB tool determines if the requested command should be blocked or if the command should be allowed.
4. If a command is blocked, the SWB tool returns to the application program without passing any command to the BIOS I/O routines. Depending on SWB tool configuration either **success** or **error** is returned for the command status.
5. If the command is allowed (not blocked), the command is passed to the BIOS and the BIOS I/O routine issues required I/O commands (ATA, SCSI or other) to the drive controller so that the desired I/O operation occurs on the hard drive.
6. Results are returned to the application program.

Figure 1-2 SWB Tool Operation



1.2.3 SWB Test Methodology

This section describes the methodology that has been developed to test interrupt 0x13 based SWB tools. First several issues are identified and then an approach is described that addresses the issues.

One simple strategy to determine the effectiveness of a SWB tool would be to install the tool, attempt an operation that should change the hard drive contents, and then examine the drive for any changes. This has the limitation that only effects of the selected command parameters are measured, not whether the actual commands are blocked. If there is no change to the hard drive then one cannot determine if the command is actually blocked or if the selected command parameters did not cause a change to the hard drive. A second limitation is that some commands, should only be executed in a *factory setting*. These are commands such as *low level formatting* or *diagnostic* commands that may have subtle and unexpected results. The proper parameter values for some commands are propriety and not easily determined. The user is cautioned that improper parameters may

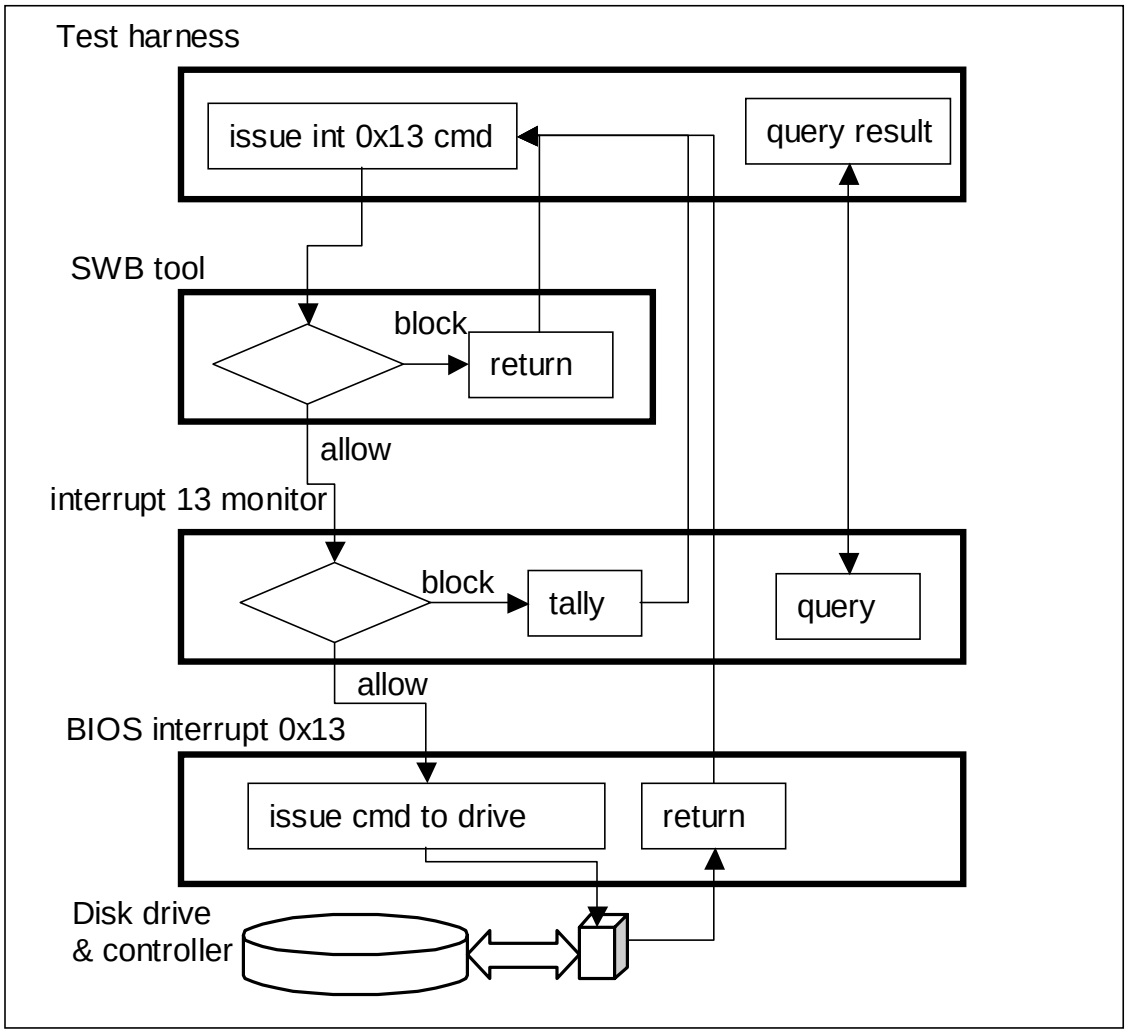
render a drive unusable. The problem is how to safely test if a SWB tool blocks such potentially destructive commands.

A more direct methodology that avoids these limitations has been developed. The normal interrupt 0x13 BIOS routine is replaced with a software monitor that counts the number of times each I/O function is called. The monitor blocks all commands so that any command can be safely issued to a SWB tool. The monitor has a secondary interface to allow a test harness to query the monitor to determine the command functions blocked or allowed by the SWB tool.

Every possible command can be tried and the results observed. First, the monitor is installed to replace the usual interrupt 0x13 processing. The monitor operates in two states: *allow command* or *block/tally*. In the *allow command* state all commands are passed to the hard drive. The *allow command* state permits the SWB tool to interact with the BIOS during installation. After the SWB tool is installed the monitor state is switched to *block/tally*. In this state, all commands passed by the SWB tool are blocked by the monitor and a tally is kept of all commands seen by the monitor. The monitor query interface allows the test harness to determine which commands are passed by the SWB tool.

Figure 1-3 illustrates the command flow during a test case. After the interrupt 0x13 monitor (tally13) and the SWB tool are installed, the test harness (test-hdl) is executed. The test harness issues every command for a given category. For example, a test of the interrupt 0x13 read command category would issue each command defined for the read command category: 0x02 (read), 0x10 (read long) and 0x42 (extended read). As each command is issued, the SWB tool intercepts the command and either blocks (return with no action) or allows the command (passes on to the interrupt 0x13 monitor). If the command gets to the interrupt 0x13 monitor a separate tally for each received command is incremented and the monitor returns to the caller (the test harness). After control returns to the test harness, the test harness queries the interrupt 0x13 monitor to get a count of the number of times the issued command has been intercepted by the monitor. If the count is zero, then the SWB tool has blocked the command. Otherwise, a non-zero count indicates that the command was not blocked.

Figure 1-3 Test Harness and Interrupt Monitor Operation



1.3 Test Case Structure

Each software write block test case is designed to follow a sequence of steps that setup the test, execute the software write block tool under test and measure the results. Most test cases follow the same test procedures. However, some test cases require a special procedure. The programs listed in Table 1-2 are required for testing.

Table 1-2 Software Required for Testing

Program	Description
SWB Tool	The software write block tool to be tested (e.g., HDL or PDBLOCK)
Monitor	The interrupt 0x13 monitor program. The monitor program blocks all interrupt 0x13 command functions, counts the number of times each function is requested for each drive, and provides an interface for retrieving those counts for each drive.

Program	Description
Test Harness	The test harness issues (requests) all interrupt 0x13 command functions for a specified command category and then queries the monitor program to determine if the function was blocked or allowed.

1.4 Software Overview

To support the capabilities necessary for testing software write block tools the programs listed in Figure 1-4 were developed.

Figure 1-4 List of Support Programs

Program	Function
Tally13	Monitor interrupt 0x13.
Test-hdl	Sent selected interrupt 0x13 commands to the SWB tool.
T-off	Turn off interrupt 0x13 monitoring.
Sig-log	Log operator observations of A/V signals

2 Requirements

This section gives a concise description of what each program is supposed to do. The section is intended to serve as a reference documenting the functionality of each program in the test harness.

2.1 Tally13

The tally13 program is used to monitor the interrupt 0x13 interface, block all commands from reaching the BIOS and count the number of times each command is received. If a SWB tool is active and a command is received by tally13 then the SWB tool does not block the given command. The tally13 program operates in either active or in passive mode. Tally13 runs in active mode to collect information on the interrupt 0x13 commands being issued by an application program. When the program is passive, all commands received are passed on to the BIOS and no commands are tallied.

Tally13 is required to log the following to stdout:

1. The program name, version number, source file creation-date, creation-time, compile-date, and compile-time.
2. The date and time program execution begins.

The specific program requirements are as follows:

3. Tally13 shall start running in passive mode.
4. There shall be a capability to switch between passive mode and active mode.
5. While in active mode all interrupt 0x13 commands shall be blocked.
6. While in active mode a count of all 0x13 commands received shall be kept.

7. There shall be a capability to query the counts of each interrupt 0x13 command received.
8. No commands shall be blocked while in passive mode.
9. No commands shall be tallied while in passive mode.
10. A unique value shall be returned via the query interface to indicate that the tool is installed.

2.2 Test-hdl

The test-hdl program is designed to operate in conjunction with the tally13 program to identify the interrupt 0x13 commands blocked or allowed by an interrupt 0x13 based software write block tool. The test-hdl program issues interrupt 0x13 commands that are then either blocked or allowed by the tool under test. Any command sent by test-hdl but not seen by tally13 has been blocked. Any command sent by test-hdl and then seen by tally13 has been allowed. The program uses the six command categories described in Table 2-1 to select the commands to issue. The command categories are described in more detail in Section 8.2.1 of *Software Write Block Tool Specification & Test Plan Version 3.0.* Any one of the six categories can be specified or all categories can be specified for testing.

Every time the program is executed, the program shall record the following to a logfile:

1. The command line (including command line options).
2. The test case ID
3. The command category to be tested.
4. The date and time program execution begins.
5. For each program source file, log the source file name, version number, source file creation date, and source file creation time.
6. The compile date & time.
7. The ID (initials or name) of the operator.
8. The name of the computer where the program is executed.
9. The number of installed drives, and the external label of each drive.

The specific program requirements are as follows (test-hdl shall …):

10. Determine if tally13 is present.
11. If tally13 is not present, issue a message and exit.
12. If tally13 is present, request tally13 to switch to active mode.
13. Before sending any commands, verify that the count value for each command to each drive is zero.
14. If any counts are non-zero before sending any commands, log the drive and command.
15. Issue each interrupt 0x13 command in the specified category.
16. For each command issued, log the command code, drive issued to, return count, status register value, carry flag setting.

17. For each drive, log the number of commands sent and the number of commands blocked.
18. For each drive, log one of the following: *{Not all | All | No} commands blocked*.
19. Verify that no commands were allowed that were not sent.

Table 2-1 Categorization of Interrupt 0x13 BIOS Commands

Categorization of Interrupt 0x13 BIOS Commands			
Command	**Code**	**Category**	**Prime Sources**
Format Track	05h	Configuration	[Gilluwe]
Format Track With Bad Sectors	06h	Configuration	[Gilluwe]
Format Cylinder	07h	Configuration	[Phoenix]
Initialize Drive Parameters	09h	Configuration	[Phoenix], [Gilluwe]
ESDI Diagnostic (PS/2)	0Eh	Configuration	[Gilluwe]
ESDI Diagnostic (PS/2)	0Fh	Configuration	[Gilluwe]
Controller RAM Diagnostic	12h	Configuration	[Gilluwe]
Drive Diagnostic	13h	Configuration	[Gilluwe]
Controller Diagnostic	14h	Configuration	[Phoenix], [Gilluwe]
Reset	00h	Control	[Phoenix], [Gilluwe]
Seek Drive	0Ch	Control	[Phoenix], [Gilluwe]
Alternate Drive Reset	0Dh	Control	[Phoenix], [Gilluwe]
Recalibrate Drive	11h	Control	[Phoenix], [Gilluwe]
Extended Seek	47h	Control	[Phoenix], [NCITS 347:2001], [Gilluwe]
Get Last Status	01h	Information	[Phoenix], [Gilluwe]
Verify Sectors	04h	Information	[Phoenix], [Gilluwe]
Read Drive Parameters	08h	Information	[Phoenix], [Gilluwe]
Test Drive Ready	10h	Information	[Phoenix], [Gilluwe]
Read Drive Type	15h	Information	[Phoenix], [Gilluwe]
Check Extensions Present	41h	Information	[Phoenix], [NCITS 347:2001], [Gilluwe]
Verify Sectors	44h	Information	[Phoenix], [NCITS 347:2001], [Gilluwe]
Get Drive Parameters	48h	Information	[Phoenix], [NCITS 347:2001], [Gilluwe]
Read Sectors	02h	Read	[Phoenix], [Gilluwe]
Read Long Sector	0Ah	Read	[Phoenix], [Gilluwe]
Extended Read	42h	Read	[Phoenix], [NCITS 347:2001], [Gilluwe]
Write Sectors	03h	Write	[Phoenix], [Gilluwe]
Write Long Sector	0Bh	Write	[Phoenix], [Gilluwe]
Extended Write	43h	Write	[Phoenix], [NCITS 347:2001], [Gilluwe]
Undefined & Unimplemented	16h-40h, 49h-FFh	Miscellaneous	

2.3 T-off

The t-off program deactivates the tally13 monitoring function by requesting that the tally13 program switch to the passive state.

Every time the program is executed, the program shall record the following to a logfile:

1. The command line (including command line options).
2. The test case ID
3. The date and time program execution begins.
4. For each program source file, log the source file name, version number, source file creation date, and source file creation time.
5. The compile date & time.
6. The ID (initials or name) of the operator.

7. The name of the computer where the program is executed.

The specific program requirements are as follows (t-off shall …):

8. Determine if tally13 is present.
9. If tally13 is not present, issue a message and exit.
10. If tally13 is present, request tally13 to switch to inactive mode.

2.4 Sig-log

The sig-log program records operator observations of audio or visual signals produced by the SWB tool under test.

Every time the program is executed, the program shall record the following to a logfile:

1. The command line (including command line options).
2. The test case ID
3. The date and time program execution begins.
4. For each program source file, log the source file name, version number, source file creation date, and source file creation time.
5. The compile date & time.
6. The ID (initials or name) of the operator.
7. The name of the computer where the program is executed.

The specific program requirements are as follows (sig-log shall …):

8. Request the user to indicate if an audio or visual signal was observed and log the response.

3 Design Notes

This section describes the design of each test harness component.

3.1 Tally13

Tally13 is written as a *terminate and stay resident* (TSR) program to monitor the interrupt 0x13 interface, block all commands from reaching the BIOS and count the number of times each command is received. After termination, a TSR program leaves a resident portion in memory to wait for a triggering event. In the case of tally13, the triggering event is use of interrupt 0x13 by an application program. The tally13 program may operate in either active or in passive mode. Usually tally13 runs in active mode to collect information on the interrupt 0x13 commands being issued by an application program. When the program is passive, all commands received are passed on to the BIOS and no commands are either blocked or tallied. Tally13 also uses interrupt 0x17 as an interface to access the counts of each interrupt 0x13 command.

The program is divided into two parts, an installation part that is executed once and a resident part that intercepts interrupts 0x13 and 0x17.

The installation part does the following:

1. Save the interrupt 0x13 vector address. This is the address of the original 0x13 routine in the BIOS. This address can be later used when tally13 is in passive mode to pass commands to the BIOS.
2. Set the interrupt 0x13 vector address to an entry point (labeled **tally_service**) in the resident part of tally13. Now any attempt to use the interrupt 0x13 disk access must go through tally13.
3. Set the interrupt 0x17 vector address to an entry point (labeled **query_service**) in the resident part of tally13. This interface is normally used to access a printer. This disables the printer service on the test machine. Since a printer is not used in the test protocol, the loss of print function has no adverse result.
4. Print a message indicating the run-date, run-time, and program version.
5. Activate the resident part by informing DOS that this is a TSR program and exit.

The resident part has two entry points, **tally_service** and **query_service**. The **tally_service** is entered whenever an interrupt 0x13 is issued by an application and the **query_service** is entered whenever an interrupt 0x17 is issued by an application. The **query_service** is used to either switch tally13 between active and passive mode or to get the count of the number of times a given command has been seen for a given drive.

The main data structure of the resident part is a two dimensional array, 256x5, of 16 bit integers. The array represents the number of times each of the 256 possible commands are seen by tally13 for each of five drives. Each integer is initially zero.

The **tally_service** interrupt (0x13) does the following:

1. The DL and AH registers are assumed to be setup as normal for interrupt 0x13 commands. In particular, DL identifies the drive and AH is the command code. The other registers are ignored.
2. If the DL does not indicate a hard drive (e.g., a floppy), the command is passed to the BIOS via the original interrupt 0x13 vector address. Hard drives are indicated by values beginning at 0x80. Floppy drives are indicated by values less than 0x80, beginning at 0x0.
3. If tally13 is in passive mode, the command is passed to the BIOS via the original interrupt 0x13 vector address.
4. Otherwise, add one to the count for the command and corresponding drive.
5. Return to caller with a status value of success.

The **query_service** entry point (0x17) does the following:

1. The AL and DH registers are assumed to be setup as normal for interrupt 0x13 commands. In particular, DL identifies the drive and AH is the command code. The other registers are ignored.
2. If the DL register is 1 or 0 then set the tally13 mode of operation to either active (DL equals one) or passive (DL equals zero) and return 0xCCFF in the AX register. The return value indicates to the calling program that tally13 is present.
3. Otherwise, return (in the CX register) the count for the command indicated in the AH register for the drive indicated in the DL register.

3.2 Test-hdl

The test-hdl program is designed to operate in conjunction with the tally13 program to identify the interrupt 0x13 commands blocked or allowed by an interrupt 0x13 based software write block tool. The test-hdl program issues interrupt 0x13 commands that are then either blocked or allowed by the tool under test. Any command sent by test-hdl but not seen by tally13 has been blocked. Any command sent by test-hdl and then seen by tally13 has been allowed. The program uses the six command categories described in Table 2-1 to select the commands to issue. Any one of the six categories can be specified or all categories can be specified for testing.

Test-hdl does the following:

1. Examine the command line parameters. If any errors are detected, a message is issued to indicate the error and the program exits.
2. Open the log file and output administrative information about the test case.
3. Use the interrupt 0x17 interface to verify that tally13 is running and switch tally13 to active mode.
4. Use the interrupt 0x17 interface to verify that all command counts are zero. If any counts are not zero issue a message identifying the command and drive involved.
5. For the specified command category and installed drives, issue each interrupt 0x13 command in the specified category, then issue the same command to the interrupt 0x17 interface and record the value returned by the interrupt 0x17 interface. If the value is zero then the command was blocked, but if the return value is not zero then the command was allowed. Log the results, including the interrupt 0x13 return code.
6. While issuing interrupt 0x13 commands keep a running total of the number of commands sent and blocked. For each drive, log the number of commands sent and the number of commands blocked.
7. Use the interrupt 0x17 interface to verify that all command counts total to an expected value such that no unaccounted for commands were issued.

3.3 T-off

T-off gets the command line parameters, logs information about the test case and then uses the interrupt 0x17 interface to deactivate tally13.

3.4 Sig-log

Sig-log gets the command line parameters, logs information about the test case and then logs the operator observations.

4 User Manual

This section describes in detail how to use each program. The organization of this section is as follows. Section 4.1 describes the expected environment, each of the remaining sections describes one component of the test harness.

4.1 System Environment

The expected operating environment is an Intel X86 (or Pentium) architecture PC running DOS with a floppy disk and at least one hard disk drive. The hard drives may use either an IDE or a SCSI interface. The system BIOS may be a legacy BIOS (does not support the interrupt 0x13 extensions) or the BIOS may support the ATA interrupt 13 extensions for large (more than 8GB) disk access.

4.2 Tally13

Tally13 is used to monitor the 0x13 interrupt commands sent by an application. Tally13 must be used in conjunction with an application such as test-hdl that sends 0x13 interrupt commands and then uses the 0x17 interrupt interface to tally13 to obtain the results.

Tally13 has no command line parameters. Tally13 prints program version and identification information that can be captured by redirecting the standard output to a log file.

4.3 Test-hdl

The command line for test-hdl has the following parameters:

```
TEST-HDL Case Host User Category DriveList
```

Table 4-1 Test-hdl Command Line Parameters

Parameter	Description
Case	The test case ID. One to eight characters suitable for a DOS file name.
Host	The name of the computer. Arbitrary length.
User	The name or initials of the test operator. Arbitrary string with no embedded spaces.
Category	A single character code indicating the category of commands to be tested. Possible values are from *Software Write Block Tool Specification & Test Plan Version 3.0*:

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>Code</th><th>Selected Command Category</th></tr>
<tr><td>c</td><td>Control commands</td></tr>
<tr><td>i</td><td>Information commands</td></tr>
<tr><td>r</td><td>Read commands</td></tr>
<tr><td>w</td><td>Write commands</td></tr>
<tr><td>x</td><td>Configuration commands</td></tr>
<tr><td>m</td><td>Miscellaneous (and undefined)</td></tr>
<tr><td>a</td><td>All commands from all categories</td></tr>
</table>
</td></tr>
<tr><td>DriveList</td><td>A list of up to 5 external drive labels for each of the hard drives installed for the test. The label is on a sticker attached to the drive.</td></tr>
</table>

Figure 4-1 is an example test-hdl log file. The line numbers in the following refer to the lines of Figure 4-1.

Line 1 is the command line used to execute test-hdl. The test case was SWB-26, the host computer was named McMillian, the operator was identified as SN. The command set was w, indicating write commands. The hard drives used were labeled 64, E3 and 1F. The hard drives are listed in order starting with the drive at drive number 0x80. The hard drive labeled 64 is referred to by the BIOS as drive 0x80, the drive labeled E3 is referred to by the BIOS as drive 0x81 and the drive labeled 1F is referred to by the BIOS as drive 0x82. This information is repeated and labeled in lines 2-11.

Lines 6-8 document the version of the program source files along with the time and date when the program was compiled. Version 1.1 of test-hdl.cpp and version 1.2 of the include file wb-defs.h were compiled to create test-hdl.exe.

Line 12 is a heading for the following tabular information (lines 13-15, 17-19 and 21-23). The first column is a sequence number (without a heading). The second column is the test case number. The third column is the command code issued in hexadecimal. The next column is the drive number (in hexadecimal) to which the command was issued. The fifth column is the action taken by the tool under test. The possible values are either Allowed (indicating the tool did not block the command) or Blocked (indicating that the tool blocked the command from reaching the BIOS and hard drive). The Stat column gives the status code returned to test-hdl for the issued command. The state of the carry flag is indicated in the column labeled Cry. The status code and the carry flag are used to report to the caller if the command executed resulted in any errors, e.g., invalid command or I/O error. The next column reports the value returned by the 0x17 interface query indicating if the command was seen by tally13. The last column contains the name of the issued command.

Lines 16, 20 and 24 give a summary of results for each drive. The summary gives the number of commands blocked and the number of commands sent. The tool under test was set to protect drive 81 and not protect drives 80 and 82 by the command `HDL 1`, where 1 indicates protection of drive 0x81. The results in the log file show that no commands

were blocked to the unprotected drives, but that for the protected drive only some of the commands were blocked (0x03 and 0x0B).

Line 25 gives a summary for the entire test of the number of commands sent, the number blocked and the number allowed.

Lines 28-32 give a summary of the commands allowed for each drive. If the total is greater than the expected value then some command was seen by tally13 that is not accounted for.

Figure 4-1 Example Test-hdl Log File

```
1.  CMD: A:\TEST-HDL.EXE SWB-26 McMillian SN w 64 E3 1F
2.  Case: SWB-26
3.  Command set: Write
4.  Date: Mon Sep 15 13:19:23 2003
5.
6.  Version: @(#) test-hdl.cpp Version 1.1 Created 08/23/03 at 10:13:51
7.      @(#) wb-defs.h Version 1.2 Created 08/31/03 at 08:18:19
8.      Compiled on Aug 31 2003 at 08:10:54
9.  Operator: SN
10. Host: McMillian
11. Number of drives 3, Drives: 64 E3 1F
12.       Case  Cmd Drv Action  Stat Cry Count Cmd Name
13.   0 SWB-26 <03> 80 Allowed 0000 Off     1  WriteSectors
14.   1 SWB-26 <0B> 80 Allowed 0000 Off     1  WriteLong
15.   2 SWB-26 <43> 80 Allowed 0000 Off     1  ExtWrite
16. Results for SWB-26 category w on drive 80 No commands blocked (0 of 3)
17.   0 SWB-26 <03> 81 Blocked 0000 Off     0  WriteSectors
18.   1 SWB-26 <0B> 81 Blocked 0000 Off     0  WriteLong
19.   2 SWB-26 <43> 81 Allowed 0000 Off     1  ExtWrite
20. Results for SWB-26 category w on drive 81 Not all commands blocked (2 of 3)
21.   0 SWB-26 <03> 82 Allowed 0000 Off     1  WriteSectors
22.   1 SWB-26 <0B> 82 Allowed 0000 Off     1  WriteLong
23.   2 SWB-26 <43> 82 Allowed 0000 Off     1  ExtWrite
24. Results for SWB-26 category w on drive 82 No commands blocked (0 of 3)
25. Summary: 9 sent, 2 blocked, 7 not blocked
26.
27.
28. Number of Commands not blocked (should total to 7)
29. Drive  Count
30.   80       3
31.   81       1
32.   82       3
```

4.4 T-off

T-off is used to switch tally13 from active mode to passive mode. The t-off command line is:

```
T-OFF Case Host User
```

Table 4-2 T-off Command Line Parameters

Parameter	Description
Case	The test case ID
Host	The name of the computer
User	The name or initials of the test operator

The log file for t-off is self explanatory.

Figure 4-2 Example T-OFF Log File

```
CMD: A:\T-OFF.EXE SWB-39 Cadfael JRL
Case: SWB-39
Date: Tue Sep 02 08:15:13 2003

Version: @(#) t-off.cpp Version 1.1 Created 08/02/03 at 16:24:48
      Compiled on Aug  2 2003 at 16:14:25
Operator: JRL
Host: Cadfael
```

4.5 Sig-log

Sig-log is used to record operator observations of audio visual signal by the SWB tool under test. The sig-log command line is:

```
SIG-LOG Case Host User
```

Table 4-3 Sig-log Command Line Parameters

Parameter	Description
Case	The test case ID
Host	The name of the computer
User	The name or initials of the test operator

The command prints a message requesting the operator's observations and then records the operator's response to the log file. The log file for sig-log is self explanatory.