NISTIR 7297-A

FS-TST 2.0: Forensic Software

Testing Support Tools

Test Plan, Test Design Specifications, and Test Case Specification

April 25, 2005

Serban I. Gavrila
VDG Inc.

NIST
Technology Administration
U.S. Department of Commerce

Abstract

This NIST Internal Report deals with Release 2.0 of a software package, Forensic Software Testing Support Tools (FS-TST 2.0), developed to aid the testing of disk imaging tools typically used in forensic investigations. The package includes programs that initialize disk drives, detect changes in disk content, and compare pairs of disks. This Internal Report consists of three parts.

This is Part A, *Test Plan, Test Design Specifications, and Test Case Specification*. It covers the planning, design, and specification of testing of FS-TST 2.0. The setup of disk drives and the testing is to be performed in the Linux environment; however, some tests will require interaction with the MS-DOS operating system.

Part B, *Test Summary Report*, is a companion document. It reports the result of testing the FS-TST 2.0 package according to Part A. Two programs might have had slightly more convenient behavior in erroneous cases, but no anomalies were found in testing.

Part C, *Code Review Report*, is an additional companion document. It covers the planning and specification of reviewing all the source code in the package and reports the results of the code reviews. Nothing was found in the code reviews that should cause invalid results, that is, that should lead to an imaging tool with systematic errors being incorrectly passed as adhering to the assertions.

The intended audience for this document should be familiar with the Linux operating system, computer operation, and computer hardware components such as hard drives.

Keywords: Computer forensic tool; disk imaging; software testing; testing support tools; FS-TST.

Table of Contents

Introduction

The Computer Forensics Tool Testing (CFTT) project at the National Institute of Standards and Technology (NIST), an agency of the United States Department of Commerce, provides a measure of confidence in the software tools used in computer forensic investigations. This document focuses on a class of tools called disk-imaging tools that copy or "image" hard disk drives. Forensic Software Testing Support Tools version 2.0 (FS-TST 2.0) is a software package that supports the testing of disk imaging tools. FS-TST 2.0 includes 10 tools that perform hard disk initialization, faulty disk simulation, hard disk comparisons, extraction of information from a hard disk, and copying of disks or disk partitions.

This document covers the planning and specification of testing the tools included in the FS-TST 2.0 package.

A portion of this work was funded by the National Institute of Justice (NIJ) through an interagency agreement with the NIST Office of Law Enforcement Standards.

Section A: FS-TST Test Plan

A1 Introduction

A1.1 Objectives

A test plan for FS-TST should support the following objectives:

(1) To detail the activities required to prepare for and conduct the testing of FS-TST.
(2) To define the sources of the information used to prepare the plan.
(3) To define the test tools and environment needed to conduct the FS-TST tests.

A1.2 Background

The Software Conformance and Diagnostics Testing (SCDT) Division of NIST has developed a software package called Forensic Software Testing Support Tools (FS-TST) version 2.0, comprising tools used in testing of disk imaging tools, which, in turn, are used in forensic investigations. Testing the FS-TST tools provides a degree of confidence in using them to test the disk imaging tools.

A1.3 Scope

The test plan covers:

(1) Testing of the functionality of FS-TST, as described in document [FST-RDU-20] - see section A1.4 References below.
(2) Testing FS-TST compliance with requirements stated in document [FST-RDU-20].

A1.4 References

The following documents were used as sources of information for the test plan:

1. FS-TST: Forensic Software Testing Support Tools. Requirements, Design Notes, and User Manual. Version 2.0, February 2005 (FST-RDU-20).
2. IEEE Standard for Software Test Documentation, IEEE Std. 829-1998 (IEEE-01).

A2 Test Items

The items to be tested are the tools included in FS-TST 2.0, namely: ***diskwipe***, ***corrupt***, ***adjcmp***, ***diskcmp***, ***partcmp***, ***logcase***, ***logsetup***, ***partab***, ***diskchg***, and ***seccmp***.

A3 Features to be tested

This section describes the features/requirements of each tool that need to be tested. Most of the support tools share common functionality. These common requirements and features are described once for all tools and then referenced as needed.

A3.1 Common functional features

A3.1.1 Hard disk drive logging

A program required to do disk logging must record the following information in the specified log file for the specified disk drive:

1. The type of the hard disk drive interface - IDE or non-IDE (SCSI).
2. The disk geometry, i.e., maximum allowed cylinder value, maximum allowed head value, number of sectors per track, and total number of sectors.
3. The disk drive model number and serial number.

A3.1.2 Program execution logging

A program required to do program execution logging must record:

1. The program name, version number, source file creation date and time, and compile date and time.
2. The support library name, version number, source file creation date and time, and compile date and time.
3. The header file name, version number, and source file creation date and time.
4. The command line (including command line options).
5. The date and time program execution begins and ends, and the elapsed time.
6. The test case ID.
7. The name of the computer where the program is executed.
8. A user supplied comment.
9. Either start a new log file or append to an existing log file.
10. Print a summary of the program command line and command line options, then exit.

A3.1.3 Partition table logging

A program required to do partition table logging must record the following information for the partition table of the specified disk drive in the specified log file:

1. For each partition table entry in the master boot record partition table and each partition table in any extended partition, print the following: starting LBA address, partition length, starting cylinder/head/sector address, ending cylinder/head/sector address, bootable flag, partition code (in hexadecimal).
2. For common partition types (FAT12, FAT16, FAT32, extended, Linux ext2, Linux swap, and NTFS) print a descriptive string, e.g., Fat32 for type code 0x0B.

A3.1.4 Comparison logging

A program that compares a source to a destination is required to do comparison logging. A source or destination is defined to be a block of contiguous disk sectors. A source or destination can be an entire disk, a disk partition, or a block of sectors located between two partitions. The source is assumed to have been initialized by ***diskwipe*** with the source fill byte, and the destination is assumed to have been initialized by ***diskwipe*** with the destination fill byte.

1. Summarize corresponding sectors of the source and destination with counts of the sectors compared, sectors matching, sectors differing and the total number of bytes that are different. Note that if large disk drives with few matching bytes are

compared, then the total number of differing bytes may exceed the maximum integer that can be represented by a variable. In this case, overflow is permitted without notification.
2. If the source and destination are not the same size, log the size of each and the difference in size.
3. If the destination is larger than the source, categorize the excess sectors according to the following: zero fill (every byte is zero), ***diskwipe***-style fill, and other contents. The ***diskwipe***-style fill is actually three categories: source fill byte, destination fill byte, and other fill byte.
4. For each category, the first few sectors belonging to the category are logged. A contiguous block of sectors is logged as a hyphen-separated pair of integers, i.e., start sector - last sector.

A3.1.5 Error reporting

The following requirements apply to all programs except as noted under each program.

1. If the command line parameters are not valid, print an error message indicating the problem, print a summary of the program command line and command line options, then exit.
2. If any I/O operation fails, print a diagnostic message and exit.
3. If any I/O operation fails, the content of the log file is undefined (the log file should be considered corrupt).

A3.2 Individual program features/requirements

A3.2.1 diskwipe features

1. Log the specified hard disk drive.
2. Log the program execution.
3. Allow specification of at least three log file names: one for a source disk, one for a destination disk, and one for a media disk.
4. Write the specified content from Table 2 of document FST-RDU-20 to each disk sector of the specified drive.
5. By default, use the number of heads obtained from the BIOS extensions; however, optionally allow specification of the number of heads to override the value from BIOS.

A3.2.2 partab features

1. Log the specified hard disk drive.
2. Log the program execution.
3. For each partition table entry in the master boot record partition table and each partition table in any extended partition, print the following: starting LBA address, partition length, starting C/H/S address, ending C/H/S address, bootable flag, partition type code (in hexadecimal).
4. For common partition types (FAT12, FAT16, FAT32, extended, Linux Ext2, Linux swap, NTFS) print a distinctive string, e.g., “Fat32” for FAT32 partitions.

5. Use a different log file name for each hard disk drive.
6. Log (optionally by command line control) a unique identification for each partition that can be used by the ***partcmp*** tool to select partitions for comparison.
7. Log (optionally by command line control) empty partition table entries.

A3.2.3 diskchg features

1. Log the specified hard disk drive.
2. Log the program execution.
3. Allow specification of disk sector addresses in either CHS or LBA format.
4. Set every byte of a specified sector to zero.
5. For a specified sector *s*, a specified address *a* (possibly not the same as the specified sector), a specified disk geometry, and a specified fill value, fill sector *s* with the contents of a ***diskwipe*** style fill using *a* as the address value for the fill. In other words, set sector *s* to the contents that ***diskwipe*** would use for the sector at location *a* on a disk with the specified geometry using the specified fill value.
6. For a specified sector, a specified offset within the sector, and a specified value, set the byte at the offset within the sector to the specified value.
7. For a specified hard drive, a specified sector, a specified offset within the sector, and a specified count *count*, log the contents of *count* bytes from the specified sector starting at the specified offset.
8. Allow interactive examination of sector contents.
9. Use a different log file name for each function.

A3.2.4 seccmp features

1. Log the specified source drive.
2. Log the specified destination drive.
3. Log the program execution.
4. If the sectors to compare are not ***diskwipe*** style filled or zero filled, log any differences between the source sector and the destination sector.
5. ***diskwipe*** style filled sectors or zero filled sectors are logged with no need for comparison.
6. Allow specification of an alternate log file name.

A3.2.5 partcmp features

1. Log the specified source drive.
2. Log the specified destination drive.
3. Log the program execution.
4. Log the comparison between the source partition and the destination partition.

A3.2.6 diskcmp features

1. Log the specified source drive.
2. Log the specified destination drive.
3. Log the program execution.
4. Log the comparison between the source drive and the destination drive.
5. If there is a read error the comparison results are undefined.

6. If there are any read errors, then continue scanning the disk and log a count of the number of tracks with read errors on each disk.

A3.2.7 corrupt features

1. Log the program execution.
2. Change a specified byte at a specified location in a specified file to a specified value.
3. Log the original value at the specified location.
4. Log the new value at the specified location.

A3.2.8 logsetup features

1. Record the following: disk label, host computer, operator, operating system loaded, date and time.

A3.2.9 logcase features

1. Record the following: Test case ID, host computer, operator, source disk drive, destination disk drive, other disk drive, date and time.

A3.2.10 adjcmp features

1. Log the specified source drive.
2. Log the specified destination drive.
3. Log the program execution.
4. Log the partition table for the specified hard drive.
5. For each disk, assign each sector to a contiguous block of sectors, called a *disk chunk*, such that each disk chunk is assigned to one of the following *chunk categories*: a sector contained within a partition, a sector contained within a partition boot track, the unallocated sectors between two partitions, or unallocated sectors after the last partition on the disk.
6. Record the location of each disk chunk in the log file.
7. Allow specification of corresponding disk chunks between the source hard drive and the destination hard drive. (A disk chunk on the source drive is compared to the corresponding disk chunk on the destination drive.)
8. Log the correspondence between source disk chunks and destination disk chunks, i.e., for each disk chunk on the source drive, log the disk chunk on the destination that the source disk is to be compared to.
9. Log the comparison between each pair of corresponding disk chunks.
10. For any destination disk chunks that have no corresponding source chunk categorize the sectors of the disk chunk according to the following: zero fill (every byte is zero), ***diskwipe*** style fill, and other contents. The ***diskwipe*** style fill is actually three categories: source fill byte, destination fill byte and any other fill byte. For each category, the first few (up to some arbitrary limit) sectors belonging to the category are logged. A contiguous block of sectors is logged as a hyphen separated pair of integers (start sector - last sector).

11. Log a summary as follows:
    - Number of boot tracks, total number of sectors assigned to boot tracks, and number of boot track sectors that do not compare equal.

- Number of partitions, total number of sectors assigned to some partition, and number of corresponding partition sectors that do not compare equal.
- Number of unallocated chunks with a corresponding unallocated chunk, number of sectors in this category and number of corresponding sectors that do not compare equal.
- Number of excess sectors in destination chunks that have a corresponding source chunk, number of sectors that have every byte set to zero, and number of remaining sectors.
- Number of sectors in destination chunks that do not have a corresponding source chunk, number of sectors that have every byte set to zero, and number of remaining sectors.
- Total number of source sectors and total number of destination sectors.

A3.2.11 sechash features

1. Compute a SHA-1 for a specified block of continuous sectors from the designated hard drive.
2. Log the computed hash value.
3. Allow the specification of at least two log file names, one for reference before a tool is run and one for comparison after a tool is run.
4. Log the specified hard drive.
5. Log the program execution

A3.2.12 diskhash features

1. Compute a SHA-1 for the designated hard drive.
2. Log the computed hash value.
3. Allow the specification of at least two log file names, one for reference before a tool is run and one for comparison after a tool is run.
4. Log the specified hard drive.
5. Log the program execution

A4 Approach

Testing personnel will develop the test cases and procedures, based on the list of features for which each tool will be tested, the applicable FS-TST documentation (FST-RDU-20), and the manner in which the tool will be used. The tools will be tested to ensure that their behavior corresponds to that outlined in the documentation. In the test cases developed, the value logged will be compared with known values acquired by other methods.

A5 Pass/fail criteria

If a tool tested does not possess one or more of the features listed for that tool, then the tool will fail the test. The tool will also fail the test if inaccuracies are found in the logs produced by that tool. Otherwise, the tool will pass the test.

A6 Test deliverables

Test documentation:
(1) FS-TST Test Plan
(2) FS-TST Test Design Specifications
(3) FS-TST Test Case Specifications
(4) FS-TST Test Summary Report

Test scripts:
(1) Scripts used to prepare the environment for and launch the test procedures.
(2) Scripts used to extract information selectively from the log files.

A7 Test tasks

Task	Predecessor Tasks
1. Prepare test plan	FS-TST design, requirements, functional specifications
2. Prepare test design specifications	Task 1
3. Prepare test case specifications	Task 2
4. Prepare test procedure specifications	Task 3
5. Obtain hardware and software required for testing the software item	Task 4
6. Execute test procedure for the software item	Task 5
7. Observe results of testing	Task 6
8. Repeat tasks 5-7 until all items have been tested	Task 7
9. Prepare test summary report	Task 8

A8 Environmental needs

A8.1 Hardware

A8.1.1 Host Computers

The following computers were available for testing:

Name	BIOS	HDD Slots
McMillan	Extended	3 IDE + 2 SCSI
Frank	Extended	2 IDE + 2 SCSI + 2 SATA

A8.1.2 Hard Disk Drives

The following hard disk drives were used for testing:

Label	Model	Interface	Sectors	GB
3B	MAG3091L SUN9.0G	SCSI	17,689,266	8
7F	MAXTOR 6L040J2	IDE	78,177,792	40
80	WDC WD800BB-00CAA1	IDE	156,301,488	80
81	WDC WD800BB-00CAA1	IDE	156,301,488	80
82	WDC WD800BB-00CAA1	IDE	156,301,488	80
CC	SEAGATE ST336705LC	SCSI	71,687,370	34
10B	WDC WD2500JD-22F	SATA	488,397,168	250

A8.2 Software

Besides the software tools being tested, a variety of other software tools are needed in order to prepare the test cases (e.g., to create partitions), or to provide a means of evaluating the test results (e.g., an alternative way of computing a disk hash). The following software was available as testing support:

Partition Magic ® Pro, Version 6.0, PowerQuest Corporation.
Disk Editor (diskedit), Version 8.0, Symantec Corporation.
Disk Editor (diskedit), Norton Utilities 2002, Symantec Corporation.
Red Hat Linux 8.2 Operating System.
Fedora Core 3 (Red Hat) Operating System.
NIST Forensics Software Testing Support Tools FS-TST 1.0 (for DOS)
NIST Computer Forensic Reference Data Sets (CFReDS) script *cal-drive.csh* (see http://www.cfreds.nist.gov/) and two variants of this script, cal-drive-count.csh and cal-drive-count-seek.csh.

Section B: FS-TST Test Design Specification

B1 *diskwipe* Test Design Specification

B1.1 Features to be tested

1. Log the specified hard disk drive (see section A3.1.1).
2. Log the program execution (see section A3.1.2).
3. Allow specification of at least three log file names: one for a source disk, one for a destination disk, and one for a media disk.
4. Write the specified content from Table 2 of document FST-RDU-20 to each disk sector of the specified drive.
5. By default, use the number of heads obtained from the BIOS extension. Optionally allow specification of the number of heads to override the value from BIOS.

B1.2 Approach refinements

Feature 1 will pass the test if and only if for each of the above test cases the C/H/S values recorded in the log file are reasonable. Note that verifying the correctness of the disk geometry will not be straightforward. Therefore, reasonable values are deemed correct.

Several test cases will be created to test that ***diskwipe*** logs the program execution correctly. The *-comment* option will be used with one-word or multi-word comments. It will also be checked that when not used, the tool will ask the user to enter a comment to be logged. A test case will verify that a log file is created when none is present, another that log records are appended when a log file is already present, and another that the old log file will be destroyed and a new file created when ***diskwipe*** is run with the *-new_log* option. We will also test the creation of a log file with a given name. Some test cases will be used to test that the *-h* option makes ***diskwipe*** print its usage mode on the stdout.

The approach to testing the third feature will be to use the three command line options *-src*, *-dst*, and *-media*, and verify that each log file name is unique.

The fourth feature will be tested over a variety of hardware configurations. The disk sector addressing method, BIOS type, and hard drive type will be varied. Several sectors from the beginning and end of the first, last, and two arbitrary cylinders will be checked for correct syntax and content using a commercial tool (e.g., ***diskedit***).

The approach to testing feature 5 will be to run ***diskwipe*** using the *-heads* option with a different number of heads than the one obtained from the BIOS.

B1.3 Test Identification

Case Id	Description/Options used	Features tested
dkw-01	-comment w	1, 2, 3, 4
dkw-02	-new_log -comment “w1 …” -noask	1, 2, 3, 4
dkw-03	-noask -dst -heads n	1, 2, 3, 4, 5
dkw-04	-noask -src	1, 2, 3, 4
dkw-05	-noask -media	1, 2, 3, 4
dkw-06	-noask -log_name x	1, 2, 3, 4
dkw-07	-noask -src -log_name x	1, 2, 3, 4
dkw-08	-noask -media -new_log -log_name x	1, 2, 3, 4
dkw-09	-serial ATA disk -new_log -noask	1, 2, 3, 4
dkw-10	No arguments, wrong arguments, -h (alone or with other options on the command line)	2

B2 *partab* Test Design Specification

B2.1 Features to be tested

1. Log the specified hard disk drive.
2. Log the program execution.
3. Log the partition table (see section A3.1.3).
4. Use a different log file name for each hard drive.
5. Log (optionally by command line control) a unique identification for each partition that can be used by the ***partcmp*** tool to select partitions for comparison.
6. Log (optionally by command line control) empty partition table entries.

B2.2 Approach refinements

Feature 1 will pass the test if and only if for each of the above test cases the C/H/S values recorded in the log file are reasonable. Note that verifying the correctness of the disk geometry will not be straightforward. Therefore, reasonable values are deemed correct.

The approach to testing feature 2 will be to run ***partab*** using different combinations of command line options and verifying that the proper information is logged in the log file. For example, we will verify that the user is prompted for a descriptive comment when running ***partab*** without the *-comment* option; also, we will verify that the comment is logged correctly when running ***partab*** with the *-comment* option followed by a single-word or multi-word comment. We will test whether partab correctly appends the log records to an existing log file, or creates a new file if the option *-new-log* is used. We will use the *-h* option to test whether *partab* displays a usage mode.

The approach to testing feature 3 will be to use the ***diskchg*** tool or another disk editor like PartitionMagic to collect the relevant partition information from the partition table(s) of the hard drive. The output of ***partab*** can then be compared with the information collected by the other tools. Testing will consist of running ***partab*** on hard disks with a variety of partition types and number of partitions. We will use partition types supported (FAT16, FAT32, extended, NTFS, Linux ext2, and Linux swap) as well as not supported by ***partab*** (e.g., HPFS). The tester will visually inspect the information logged by ***partab***.

To test for uniqueness in log file names (feature 4), we will run ***partab*** on hard drives with different interfaces mounted as devices with different names (e.g., /dev/hdb, /dev/sda) and we will inspect the names of the log file created for each hard drive for uniqueness.

The approach to testing for uniqueness of partition identifiers (feature 5) will be to run ***partab*** on hard disk drives with multiple primary and/or logical partitions. The tester will visually check that the log file created contains entries for each of the partitions and that for each partition there is assigned a unique identifier. This test will be performed with and without the *-all* option to determine that unique identifiers are assigned when extended partition entries are logged as well as when they are not.

The approach to testing logging of empty partition table entries will be to run ***partab*** on hard drives with various numbers of primary and logical partitions that have or do not have empty entries, and ensure that ***partab*** correctly logs them.

B2.3 Test Identification

Case Id	Partitions	Description/Options	Features
ptb-01	None	-all -comment w	1, 2
ptb-02	-primary FAT	-all -comment "w1…" -new_log	1, 2, 3, 4, 5, 6
ptb-03	-primary FAT32	-all -interactive comment -append log	1, 2, 3, 4, 5, 6
ptb-04	-primary NTFS	-all -log_name x -interactive comment	1, 2, 3, 4, 5, 6
ptb-05	-primary FAT32 huge -primary Linux ext2 -primary Linux swap	-all -log_name x	1,2,3,4,5,6
ptb-06	-primary FAT -primary FAT32 hidden -primary HPFS	-all -new_log -log_name x	1,2,3,4,5,6
ptb-07	-multiple extended and logical partitions	-all -new_log	1,2,3,4,5,6
ptb-08		-no arguments -or incorrect syntax -or -h alone -or -h with other options	2

B3 *diskchg* Test Design Specification

B3.1 Features to be tested

1. Log the specified hard disk drive.
2. Log the program execution.
3. Allow specification of disk sector addresses in either CHS or LBA format.
4. Set every byte of a specified sector to zero.
5. For a specified sector *s*, a specified address *a* (possibly not the same as the specified sector), a specified disk geometry, and a specified fill value, fill sector *s* with the contents of a ***diskwipe*** style fill using *a* as the address value for the fill. In other words, set sector *s* to the contents that ***diskwipe*** would use for the sector at location *a* on a disk with the specified geometry using the specified fill value.
6. For a specified sector, a specified offset within the sector, and a specified value, set the byte at the offset within the sector to the specified value.
7. For a specified hard drive, a specified sector, a specified offset within the sector, and a specified count *count*, log the contents of *count* bytes from the specified sector starting at the specified offset.
8. Allow interactive examination of sector contents.
9. Use a different log file name for each function.

B3.2 Approach refinements

Feature 1 will pass the test if and only if for each of the above test cases the C/H/S values recorded in the log file are reasonable. Note that verifying the correctness of the disk geometry will not be straightforward. Therefore, reasonable values are deemed correct.

The approach to testing feature 2 will be to run ***diskchg*** using most combinations of command line options and verifying that the proper information is logged in the log file.

The approach to testing feature 3 will be to run ***diskchg*** using both CHS and LBA sector addresses for each function, and observing whether the results are identical.

The approach to testing feature 4 will be to run ***diskchg*** using the -zero option, then using ***diskedit*** and/or ***diskchg*** itself to examine the bytes of the specified sector.

The approach to testing feature 5 will be to run ***diskchg*** using the -fill option, and specifying different combinations of sector addresses, fill addresses, disk geometries (zero and non-zero number of heads), and fill values. The resulting sector contents will be compared with the contents of the sector at the fill address as written by ***diskwipe***, using ***diskedit*** and/or ***diskchg*** itself to display those contents.

The approach to testing feature 6 will be to run ***diskchg*** using the -write option for different sector addresses, offsets, and values, then examining the byte at that offset within the specified sector by using ***diskedit*** and/or ***diskchg*** itself.

The approach to testing feature 7 will be to run ***diskchg*** using the -read option for different sector addresses, offsets, and counts, then comparing the logged results with the values displayed by ***diskedit***.

The approach to testing feature 8 will be to run ***diskchg*** using the -exam option, entering different sector addresses when prompted, and comparing the logged results with those displayed by ***diskedit*** and/or those displayed by the function /read of ***diskchg*** itself.

The approach to testing feature 9 will be to check whether the name of the log file produced by ***diskchg*** for each of the functions -read, -exam, -fill, -write, -zero, is unique for that function.

B3.3 Test Identification

Case Id	Description/Options	Features
dch-01	-delete all log files -comment w -exam using LBA addresses of arbitrary sectors including the first and last -exam using CHS addresses of same sectors	1, 2, 3, 8, 9
dch-02	-append log records to log file created in previous case by not using -new_log -comment “w1 …” -exam using LBA/CHS addresses of arbitrary sectors	1, 2, 3, 8, 9
dch-03	-new_log -use interactive comment -exam using LBA/CHS address of sectors outside the target disk range	1, 2, 3, 8, 9
dch-04	-read using LBA address of an arbitrary sector	1, 2, 3, 7, 9
dch-05	-new_log -read using CHS address of same sector with offset too large	1, 2, 3, 7, 9
dch-06	-new_log -read using LBA/CHS with length too large	1, 2, 3, 7, 9
dch-07	-new_log -read using LBA/CHS with offset+length too large	1, 2, 3, 7, 9
dch-08	-new_log -read using LBA/CHS outside disk range	1, 2, 3, 7 9
dch-09	-new_log -fill using CHS and detected geometry (heads 0) -read same sector	1, 2, 3, 5, 9
dch-10	-new_log -fill using LBA of same sector and detected geometry (heads = detected number of heads/sector)	1, 2, 3, 5, 9

dch-11	-new_log -fill using LBA of same sector, a new geometry, and another value -read or -exam same sector	1, 2, 3, 5, 9
dch-12	-new_log -write using LBA -read or -exam same sector	1, 2, 3, 6, 9
dch-13	-new_log -write using CHS of same sector -read or -exam sector	1, 2, 3, 6, 9
dch-14	-new_log -write using LBA or CHS, offset too large	1, 2, 3, 6, 9
dch-15	-new_log -write using LBA or CHS of sector outside disk range	1, 2, 3, 6, 9
dch-16	-zero using LBA of first sector -read or -exam sector	1, 2, 3, 4, 9
dch-17	-log_name -zero using CHS of last sector -read or -exam sector	1, 2, 3, 4, 9
dch-18	-log_name with same name as before -zero using LBA of arbitrary sector -read or -exam sector	1, 2, 3, 4, 9
dch-19	-log_name with same name as before -new_log -zero using LBA or CHS of sector outside disk range	1, 2, 3, 4, 9
dch-20	-h	2

B4 *seccmp* Test Design Specification

B4.1 Features to be tested

1. Log the specified source drive.
2. Log the specified destination drive
3. Log the program execution.
4. If the sectors to compare are not ***diskwipe***-style filled or zero filled, log any differences between the source sector and the destination sector.
5. ***diskwipe***-style filled sectors or zero filled sectors are logged with no need for comparison.
6. Allow specification of an alternate log file name.

B4.2 Approach refinements

Features 1 and 2 will pass the test if and only if for each of the above test cases the C/H/S values recorded in the log file are reasonable. Note that verifying the correctness of the disk geometry is not straightforward. Therefore, reasonable values are deemed correct.

The approach to testing feature 3 will be to run ***seccmp*** using different combinations of command line options and verifying that the proper information is logged in the log file.

The approach to testing feature 4 will be to run ***seccmp*** using source and destination sectors that are not both diskwipe-style filled or both zero, and checking whether the differences are logged.

The approach to testing feature 5 will be to run ***seccmp*** using source and destination sectors that are both diskwipe style filled or zero filled, and check whether they are logged without comparison.

The approach to testing feature 6 will be to run ***seccmp*** using the *-log_name* option followed by an alternate log file name.

B4.3 Test Identification

Case Id	Description/Options	Features
scm-01	-comment w -compare first disk sectors, not diskwipe- or zero-filled	1, 2, 3, 4
scm-02	-append log records -comment “w1 …” -compare last disk sectors, diskwipe-filled	1, 2, 3, 5
scm-03	-new_log -try comparing sectors outside range	1, 2, 3
scm-04	-new_log -same source fill value and destination fill value -interactively enter sector addresses	1, 2, 3, 4, 5

	-sectors diskwipe-filled, all combinations of real fill values -when real source fill value equals real destination fill value, consider sectors with same or different headers	
scm-05	-new_log -different source fill value and destination fill value -interactively enter sector addresses -sectors diskwipe-filled, all combinations of real fill values -when real source fill value equals real destination fill value, consider sectors with same or different headers	1, 2, 3, 4, 5
scm-06	-log_name -interactively enter sector addresses -combinations of diskwipe/zero, diskwipe/not diskwipe and not zero, zero/zero, zero/not diskwipe and not zero, for source/destination sectors	1,2, 3, 4, 5, 6
scm-07	-h	3

B5 *partcmp* Test Design Specification

B5.1 Features to be tested

1. Log the specified source drive.
2. Log the specified destination drive.
3. Log the program execution.
4. Log the comparison between the source partition and the destination partition.

B5.2 Approach refinements

Feature 1 will pass the test if and only if for each of the above test cases the C/H/S values recorded in the log file are reasonable. Note that verifying the correctness of the disk geometry will not be straightforward. Therefore, reasonable values are deemed correct.

The approach to testing feature 3 will be to run ***partcmp*** using different combinations of command line options and verifying that the proper information is logged in the log file.

The approach to testing feature 4 will be to run ***partcmp*** using various source and destination partitions, with different or equal sizes and the same or different contents, on hard drives with various interfaces, and checking the reported differences against the known ones. In general, in the setup of each test case, we will copy the smaller partition onto the larger one and modify a few predetermined sectors of the copy.

B5.3 Test Identification

Case Id	Description/Options	Features
pcm-01	-compare big primary FAT32 partitions -src is smaller than dst -same contents on the smaller length -comment w -interactive partition selection.	1, 2, 3, 4
pcm-02	-compare big primary FAT32 -src is bigger than dst -src, dst have almost same contents on the smaller length -new_log -comment “w1 …” -select -boot	1, 2, 3, 4
pcm-03	-compare primary Linux Ext2 partitions -src is bigger than dst -same contents on the smaller length -comment “w1 …” -interactive partition selection -append the log records -interactive comment	1, 2, 3, 4

	-boot	
pcm-04	-compare logical FAT32 partitions -src, dst have same size and contents -use alternate log file name (-log_name) -interactive selection -interactive comment -boot	1, 2, 3, 4
pcm-05	-compare logical FAT32 partitions -src, dst have the same size -src, dst have almost same contents -append log records to log file with alternate name -interactive selection -interactive comment -boot	1, 2, 3, 4
pcm-06	-compare logical FAT16 partitions -src is smaller than dst -same contents on the smaller length -new log file with alternate name is created when -log_name and -new_log are both used -interactive selection -interactive comment -boot	1, 2, 3, 4
pcm-07	-select with partition index pointing to empty entries	1, 2, 3, 4
pcm-08	-select with invalid partition indexes	1, 2, 3, 4
pcm-09	-h option in various ways.	1, 2, 3, 4

B6 *diskcmp* Test Design Specification

B6.1 Features to be tested

1. Log the specified source drive.
2. Log the specified destination drive.
3. Log the program execution.
4. Log the comparison between the source drive and the destination drive.
5. If there is a read error the comparison results are undefined.
6. If there are any read errors, then continue scanning the disk and log a count of the number of tracks with read errors on each disk.

B6.2 Approach refinements

Feature 1 will pass the test if and only if for each of the above test cases the C/H/S values recorded in the log file are reasonable.

The approach to testing feature 3 will be to run ***diskcmp*** using different combinations of command line options and verifying that the proper information is logged in the log file.

The approach to testing feature 4 will be to run ***diskcmp*** using various models and sizes of source and destination hard disk drives, whose contents before comparison is known, and checking the reported differences against the known ones. In general, the drives will be prepared for comparison by copying the smaller one onto the bigger one, and modifying sectors at predetermined addresses.

B6.3 Test Identification

Case Id	Description/Options	Features
dcm-01	-source disk size > destination disk size -same contents on the smaller size -comment w -no log file present.	1, 2, 3, 4
dcm-02	-source disk size < destination disk size -*almost* same contents on the smaller size -comment “w1 …” -append the log records	1, 2, 3, 4
dcm-03	-same size source and destination disks -diskwipe-style filled with same value, same geometry -a few different sectors -new_log -interactive comment.	1, 2, 3, 4
dcm-04	-same size source and destination disks -diskwipe-style filled with different values -a few equal sectors -alternate log file name using -log_name	1, 2, 3, 4
dcm-05	-h	1, 2, 3, 4, 5, 6

B7 *corrupt* Test Design Specification

B7.1 Features to be tested

1. Log the program execution.
2. Change a specified byte at a specified location in a specified file to a specified value.
3. Log the original value at the specified location.
4. Log the new value at the specified location.

B7.2 Approach refinements

The approach to testing feature 1 will be to run ***corrupt*** using different combinations of command line options and verifying that the proper information is logged in the log file. We will launch ***corrupt*** with and without the *-comment* option, to verify that it accepts and logs one-word comments, multi-word comments, and interactively entered comments. We will launch ***corrupt*** with and without the *-new_log* option, to verify that the tool creates a new, default log file, or appends the log records to an existing one. Also, we will test whether ***corrupt*** displays its usage mode when prompted by the *-h* option.

Regarding features 2, 3, and 4, we will specify valid offsets in the image file, and observe whether ***corrupt*** alters the byte at the specified offset and logs the original and new value. To test that the tool only alters the desired byte, we will make a reference copy of the image file, then run ***corrupt***, and then compare the modified image file to the reference copy. We will use the Linux command *cmp* to perform the comparison. We will also specify invalid offsets and observe whether ***corrupt*** detects the invalid offset.

B7.3 Test Identification

Case Id	Description	Features
cor-01	-alter first byte of an image file -comment -w	1, 2, 3, 4
cor-02	-alter the last byte of an image file -comment “w1…” -append the log to the existing.	1, 2, 3, 4
cor-03	-alter a byte of an image file -new_log.	1, 2, 3, 4
cor-04	-alter a byte of an image file -log_name.	1, 2, 3, 4
cor-05	-specify an offset outside the image file range. -new_log.	1, 2
cor-06	-h	1, 2, 3, 4

B8 *logsetup* Test Design Specification

B8.1 Features to be tested

1. Record the following: disk label, host computer, operator, operating system loaded, date and time.

B8.2 Approach refinements

The approach to testing feature 1 will be to run ***logsetup*** using arguments as specified in the FS-TST Version 2.0 documentation and observe whether the information provided through the command line arguments plus the current date and time extracted from the OS are correctly logged.

B8.3 Test Identification

Case Id	BIOS	Disk type	Description	Features
lgs-01	N/A	N/A	Run ***logsetup*** with 4 string arguments: the hard disk drive, the host computer, operator, OS.	1

B9 *logcase* Test Design Specification

B9.1 Features to be tested

1. Record the following: Test case ID, host computer, operator, source disk drive, destination disk drive, other disk drive, date and time.

B9.2 Approach refinements

The approach to testing feature 1 will be to run ***logcase*** using arguments as specified in the FS-TST Version 2.0 documentation and observe whether the information provided through the command line arguments plus the current date and time extracted from the OS are correctly logged.

B9.3 Test Identification

Case Id	BIOS	Disk type	Description	Features
Lgc-01	N/A	N/A	Run ***logcase*** with 6 string arguments: test case ID, the host computer, operator, source disk, destination disk, media disk.	1

B10 *adjcmp* Test Design Specification

B10.1 Features to be tested

1. Log the specified source drive.
2. Log the specified destination drive.
3. Log the program execution.
4. Log the partition table for the specified hard drive.
5. For each disk, assign each sector to a contiguous block of sectors, called a *disk chunk*, such that each disk chunk is assigned to one of the following *chunk categories*: a sector contained within a partition, a sector contained within a partition boot track, the unallocated sectors between two partitions, or unallocated sectors after the last partition on the disk.
6. Record the location of each disk chunk in the log file.
7. Allow specification of corresponding disk chunks between the source hard drive and the destination hard drive. (A disk chunk on the source drive is compared to the corresponding disk chunk on the destination drive.)
8. Log the correspondence between source disk chunks and destination disk chunks, i.e., for each disk chunk on the source drive, log the disk chunk on the destination that the source disk is to be compared to.
9. Log the comparison between each pair of corresponding disk chunks.
10. For any destination disk chunks that have no corresponding source chunk categorize the sectors of the disk chunk according to the following: zero fill (every byte is zero), ***diskwipe*** style fill, and other contents. The ***diskwipe*** style fill is actually three categories: source fill byte, destination fill byte and any other fill byte. For each category, the first few (up to some arbitrary limit) sectors belonging to the category are logged. A contiguous block of sectors is logged as a hyphen separated pair of integers (start sector - last sector).
11. Log a summary as follows:
    - Number of boot tracks, total number of sectors assigned to boot tracks, and number of boot track sectors that do not compare equal.
    - Number of partitions, total number of sectors assigned to some partition, and number of corresponding partition sectors that do not compare equal.
    - Number of unallocated chunks with a corresponding unallocated chunk, number of sectors in this category and number of corresponding sectors that do not compare equal.
    - Number of excess sectors in destination chunks that have a corresponding source chunk, number of sectors that have every byte set to zero, and number of remaining sectors.
    - Number of sectors in destination chunks that do not have a corresponding source chunk, number of sectors that have every byte set to zero, and number of remaining sectors.
    - Total number of source sectors and total number of destination sectors.

B10.2 Approach refinements

Features 1 and 2 will pass the test if and only if for each of the above test cases the C/H/S values recorded in the log file are reasonable. Note that verifying the correctness of the disk geometry is not straightforward. Therefore, reasonable values are deemed correct.

The approach to testing feature 3 will be to run ***adjcmp*** using different combinations of command line options and different layouts of the source and destination disks and verifying that the proper information is logged in the log file.

The approach to testing feature 4 will be to run ***adjcmp*** using different layouts of the source and destination disks and verifying that the partition map logged by ***adjcmp*** is identical to the one indicated by a tool like PartitionMagic.

The approach to testing features 5 and 6 will be to run ***adjcmp*** using different disk layouts, and verifying that ***adjcmp*** correctly distinguishes and categorizes the disk chunks according to the definition of a disk chunk in feature 5. Also, that ***adjcmp*** logs correctly the location of each chunk. We will use a tool like PartitionMagic to identify the chunks independently.

The approach to testing feature 7 is to run ***adjcmp*** using the -assign option on the command line and to observe whether ***adjcmp*** allows the user to interactively assign source chunks to destination chunks.

The approach to testing feature 8 is to run ***adjcmp*** using automatic or interactive chunk assignment on different disk layouts and observing whether the chunk assignment is correctly reported.

The approach to testing feature 9 is to run ***adjcmp*** using corresponding source and destination chunks whose characteristics are known (for example, with the sector contents known, being set up a priori by using a tool like ***diskchg*** or a commercial disk editor), then comparing the report to known statistics about the chunks.

The approach to testing feature 10 is to set up the disk layouts such that the destination disk has chunks that do not correspond to any source chunk, then set up the sector contents of such a chunk using ***diskwipe*** or ***diskchg*** or a commercial disk editor. Then run ***adjcmp*** and compare the report about that destination chunks with what we already know about them.

The approach to testing feature 11 is to examine the ***adjcmp*** report and to compare the summary to data about the disks and disk chunks extracted from other information sources, such as PartitionMagic, disk editors, ***diskchg***, or ***diskwipe***.

B10.3 Test Identification

Case Id	Description	Features
acm-01	-create multiple primary and logical partitions on each disk -delete all previous log files -use -comment with one-word comment -use the -layout option.	1-6
acm-02	-use -new_log to test creation of a new log file. -use -comment with a multi-word comment -use automatic assignment of disk chunks -equal partitions -in-excess destination chunks.	1-11
acm-03	-append the log records to existing log file -enter comment interactively -use -assign for manual assignment of disk chunks.	1-11
acm-04	-use -log_name to create a log file with alternate name -in-excess source chunks -partitions have well-determined different sectors..	1-11
acm-05	-use large primary/logical partitions on source and destination disks -use -new_log -use manual assignment (-assign) of source U chunks to a “don’t care” destination chunk -use source and destination chunks with src > dst and src < dst.	1-11
acm-06	-use -h option to display the usage mode.	3

B11 *sechash* Test Design Specification

B11.1 Features to be tested

1. Compute a SHA-1 for a specified block of continuous sectors from the designated hard drive.
2. Log the computed hash value.
3. Allow the specification of at least two log file names, one for reference before a tool is run and one for comparison after a tool is run.
4. Log the specified hard drive.
5. Log the program execution.

B11.2 Approach refinements

The approach to testing feature 1 and 2 is to use another application to compute the SHA-1 hash of the specified sector block and to verify that ***sechash*** writes the correct hash valued in the log file.

The approach to testing feature 3 is to run ***sechash*** with the options *-before*, *-after*, and *-log_name*, and to check whether it creates log files with different names for each option.

Feature 4 will pass the test if and only if for each of the above test cases the C/H/S values recorded in the log file are reasonable. Note that verifying the correctness of the disk geometry will not be straightforward. Therefore, reasonable values are deemed correct.

The approach to testing feature 5 will be to run ***sechash*** using different combinations of command line options, including various start and end sector addresses, and verifying that the proper information is logged in the log file.

B11.3 Test Identification

In this table, N is the last sector of the disk.

Case Id	Description/Options	Features
shs-01	-comment w -before	1, 2, 3, 4, 5
shs-02	-comment “w1 …” -before -first 0 -last N -hash md5sum	1, 2, 3, 4, 5
shs-03	-new_log -after -first 0 -last N -hash sha1sum	1, 2, 3, 4, 5
shs-04	-after -first 0 -last N	1, 2, 3, 4, 5

	-hash sha1sum	
shs-05	-log_name <name> -after -first 0 -last 0 -hash sha1sum	1, 2, 3, 4, 5
shs-06	-log_name <name> (same as before) -new_log -first 0 -last 0 -hash md5sum	1, 2, 3, 4, 5
shs-07	-before -new_log -first N -last N	1,2, 3, 4, 5
shs-08	-before -new_log -first N -last N -hash md5sum	1, 2, 3, 4, 5
shs-09	-before -new_log -first m (with 0 <= m) -last k (with m < k <= N)	1, 2, 3, 4, 5
shs-10	-before -new_log -first m (with 0 <= m) -last k (with m < k <= N)	1,2, 3, 4, 5
shs-11	-before -new_log -first m (with 0 <= m <= N) -last k (with m > k)	2
shs-12	-before -new_log -first m (with m > N) -last k (with k > m)	2
shs-13	-before -new_log -first m (with m <= N) -last n (with n > N)	2
shs-14	-h	5

B12 *diskhash* Test Design Specification

B12.1 Features to be tested

1. Compute a SHA-1 for the designated hard disk drive.
2. Log the computed hash value.
3. Allow specification of at least two log file names, one for reference before a tool is run, and one for comparison after a tool is run.
4. Log the specified hard drive.
5. Log the program execution.

B12.2 Approach refinements

The approach to testing feature 1 and 2 is to use another application to compute the SHA-1 hash of the specified hard disk drive and to verify that ***diskhash*** writes the correct hash valued in the log file.

The approach to testing feature 3 is to run ***diskhash*** with the options -before, -after, and -log_name, and to check whether it creates log files with different names for each option.

Feature 4 will pass the test if and only if for each of the above test cases the C/H/S values recorded in the log file are reasonable. Note that verifying the correctness of the disk geometry will not be straightforward. Therefore, reasonable values are deemed correct.

The approach to testing feature 5 will be to run ***diskhash*** using different combinations of command line options, including various start and end sector addresses, and verifying that the proper information is logged in the log file.

B12.3 Test Identification

Case Id	Description/Options	Features
dhs-01	-comment w -before	1, 2, 3, 4, 5
dhs-02	-comment “w1 …” -before	1, 2, 3, 4, 5
dhs-03	-new_log -before	1, 2, 3, 4, 5
dhs-04	-after	1, 2, 3, 4, 5
dhs-05	-log_name <name>	
dhs-06	-log_name <name> (same as before)	3, 4, 5
dhs-07	-log_name <name> (same as before) -new_log	1,2, 3, 4, 5
dhs-08	-h	5

B13 Disk Logging Test Design Specification

B13.1 Features to be tested

1. The disk geometry, i.e., maximum allowed cylinder value, maximum allowed head value, number of sectors per track and total number of sectors.
2. The model number and serial number.
3. The disk interface IDE/non-IDE.

B13.2 Approach refinements

The approach to testing that hard disk drives are being logged correctly will be to run relevant FS-TST tools on hard disk drives of various types (IDE, SCSI, SATA) and models and observe whether the tools record the correct information about the hard disk in the log file.

B13.3 Test Identification

Most FS-TST tools must log one or more hard disk drives. Consequently, we selected several test cases from the tools' test design specifications.

Case Id	**Disk type**	**Features**
Dkw-01	SCSI	1, 2, 3
Dkw-04	IDE	1, 2, 3
Dkw-09	SATA	1, 2, 3

B13.4 Feature pass/fail criteria

Feature 1 will pass the test if and only if for each of the above test cases the C/H/S values recorded in the log file are reasonable. Note that we expect that verifying the correctness of the disk geometry will not be straightforward. Therefore, reasonable C/H/S values will be deemed correct.

Feature 2 will pass the test if and only if in each test case, the recorded model and serial number are identical to the ones provided by the manufacturer.

Feature 3 will pass the test if and only if in all cases the chosen tool detects the disk interface correctly.

Section C: FS-TST Test Case Specifications

Rather than giving complete setup information for each test case, many test cases rely on conditions created by a preceeding test case. These are documented in the test case dependencies. The test cases may be run in any order that respects the dependencies.

C1 *diskwipe* Test Case Specifications

C1.1 Dkw-01

C1.1.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4. We will test:
- Whether ***diskwipe*** correctly displays a summary of the command line arguments and options.
- Whether data about the program, support libraries if any, and header files if any (name, version, creation, compile, and execution times), are correctly displayed.
- Whether the hard disk drive we select to be wiped is correctly logged (type of interface IDE/non-IDE, geometry, model and serial numbers).
- Whether ***diskwipe*** creates a new log file on the log disk with the default name for a destination disk.
- Whether the one-word comment supplied on the command line with the *-comment* option is logged correctly.
- Whether all other required information (see section A3.1.2) is correctly logged.
- Whether the hard disk is wiped out correctly, by displaying sectors from the beginning, middle, and the end of the disk using a commercial tool like diskedit, or the -exam function of the ***diskchg*** tool.

C1.1.2 Test setup

On the computer used for testing, mount the hard disk with the Linux OS. We assume that the FS-TST 2.0 tools reside on the same disk and that the same disk will be used as log disk.
Mount the test hard disk in a slot of the test computer. We assume that the test hard disk is a SCSI disk externally labeled “CC”, and that the slot we selected is associated with the Linux device /dev/sda. If you use other disk/slot, modify the ***diskwipe*** command line accordingly.
Boot up to Linux and delete all log files from the current directory.

C1.1.3 Test case dependencies

None.

C1.1.4 Procedure

Run ***diskwipe*** with the mandatory arguments and with the option *-comment* <comment>, where <comment> is one-word comment:

```
diskwipe dkw-01 mcmillan serban /dev/sda CC -comment
Wipeout
```

Use the *ls* command and a text editor to examine the log file's existence, prescribed name, and contents.
Use ***diskedit*** or the -exam function of the ***diskchg*** tool to examine the sectors of the wiped hard disk. (Note: you need to reboot in MS-DOS in order to run the disk editor tool).

C1.1.5 Expected results

The user is asked for confirmation before wiping the disk.
A log file named "wipedlog.txt" is created on the log disk in the current directory (by default, ***diskwipe*** should consider the disk as a destination disk, hence the "d" in the file name). The comment is logged.
The log file contains the correct information required by features 1 and 2.
The first few sectors and the last few sectors of the first cylinder, the first few sectors of the second cylinder, the first few sectors and the last few sectors of the last cylinder on the wiped disk have the required format (see Table 2 of the FST-RDU-20 document).

C1.2 Dkw-02

C1.2.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 of the diskwipe tool, as in the case dkw-01, with the following differences for feature 2:
- We will test whether ***diskwipe*** creates a new log file with the name specific to a destination disk when we specify the *-new_log* option, even though a log file with the same name already exists.
- We will test whether a multi-word comment supplied on the command line with the *-comment* option is correctly logged.
We also will test the effect of the -noask option on the program execution.

C1.2.2 Test setup

Use the test setup of case dkw-01.

C1.2.3 Test case dependencies

Dkw-01 (in order to have its log file present on the log disk).

C1.2.4 Procedure

Run ***diskwipe*** using the options *-new_log*, *-comment* followed by a multi-word comment (not containing quotes, but included in quotes), and -noask:

```
diskwipe dkw-02 mcmillan serban /dev/sda CC -new_log -
comment "Wiping a disk with 0xCC" -noask
```

Use the *ls* command and a text editor to examine the log file's existence, prescribed name, and contents.
Use ***diskedit*** or the -exam function of the ***diskchg*** tool to examine the wiped hard disk's sectors. (Note: you need to reboot in MS-DOS in order to run the disk editor tool.)

C1.2.5 Expected results

The user is no longer prompted for confirmation before wiping the disk.
The old log file “wipedlog.txt” is deleted and a new log file with the same name is created on the log disk in the current directory.
The comment is logged correctly.
The log file contains the correct information required by feature 2.
The first few sectors and the last few sectors of the first cylinder, the first few sectors of the second cylinder, the first few sectors and the last few sectors of the last cylinder on the wiped disk have the required format (see Table 2 of the FST-RDU-20 document).

C1.3 Dkw-03

C1.3.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 of the diskwipe tool. Specifically, in this case we will test:
- Whether ***diskwipe*** prompts the user for a comment when no comment is supplied on the command line.
- Whether ***diskwipe*** appends the log records to the log file created in dkw-02; we will designate the hard disk drive explicitly as a destination drive by using the *-dst* option, and we will force appending the log by *not* using the *-new_log* option.
- Whether ***diskwipe*** correctly fills the sectors according to new disk geometry specified by the *-heads* option.

C1.3.2 Test setup

Use the setup of dkw-01.

C1.3.3 Test case dependencies

Dkw-02. Do not delete the log file created in the case dkw-02, in order to append the log to it.

C1.3.4 Procedure

Run ***diskwipe*** using the *-dst*, -noask, *-heads* options:

```
diskwipe dkw-03 mcmillan serban /dev/sda CC -dst -noask -heads 200
```

Use the *ls* command and a text editor to examine the log file’s existence (and name), and contents.
Use ***diskedit*** or the -exam function of the ***diskchg*** tool to examine the wiped hard disk’s sectors.

Computing the C/H/S address of a sector in a given geometry from the LBA address of that sector may help the user to check whether ***diskwipe*** correctly filled the sectors. Here is the algorithm to convert LBA to C/H/S:

S = LBA % 63 + 1;
T = LBA / 63;

H = T % heads;
C = T / heads;

where *heads* is the number of heads per cylinder. The algorithm assumes that each track has 63 sectors. As an example, the sector 6/1/1 in the geometry with heads=255 has the LBA = 6 * 255 * 63 + 1 * 63 + 1 - 1 = 96453. Let's compute its C/H/S address for a new geometry with heads = 200:

S = 96453 % 63 + 1 = 1;
T = 96453 / 63 = 1531;
H = 1531 % 200 = 131;
C = 1531 / 200 = 7.

Thus, sector 6/1/1 in a geometry with 255 heads and 63 sectors per track becomes sector 7/131/1 in a geometry with 200 heads and 63 sectors per track.

Use the -exam or -read function of the ***diskchg*** tool with the LBA=96453. ***diskchg*** will display the actual C/H/S address (it should be 6/1/1) and the contents of that sector. Check whether the sector's header contains the values 7/131/1 and 96453 (in the format prescribed by Table 2 of the tool specifications document.)

C1.3.5 Expected results

The user is prompted for a comment. The user is not prompted for confirmation.
The log is appended to the log file "wipedlog.txt" created in the previous case.
The comment is logged.
The log file contains the correct information required by features 1, 2.
The sectors have the required format (see Table 2 of the FST-RDU-20 document), where the sector header uses the new geometry for the CHS part.

C1.4 Dkw-04

C1.4.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4, and especially whether diskwipe creates a log file with a special name for a source hard disk.

C1.4.2 Test setup

Similar to the setup of case dkw-01, but we will use an IDE disk (labeled "7F") as source hard disk drive and assume that it will be recognized as /dev/hdb in the Linux system. If you use other hard drive slot, modify the command line accordingly.

C1.4.3 Test case dependencies

None.

C1.4.4 Procedure

Run ***diskwipe*** with the *-src* and -noask options:

```
diskwipe dkw-04 mcmillan serban /dev/hdb 7F -src -noask
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents.
Use ***diskedit*** or the -exam function of the ***diskchg*** tool to examine the wiped hard disk's sectors.

C1.4.5 Expected results

The user is prompted for a comment, but not for confirmation.
A new log file "wipeslog.txt" is created on the log disk.
The comment is logged correctly.
The log file contains the correct information required by features 2.
The sectors are correctly filled.

C1.5 Dkw-05

C1.5.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4, and especially whether diskwipe creates a log file with a special name for a media hard disk.

C1.5.2 Test setup

Use the setup of *dkw-04*.

C1.5.3 Test case dependencies

None.

C1.5.4 Procedure

Run ***diskwipe*** using the options -noask and -*media*:

```
diskwipe dkw-05 mcmillan serban /dev/hdb 7F -noask -media
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents.
Use ***diskedit*** or the -exam function of the ***diskchg*** tool to examine the wiped hard disk's sectors.

C1.5.5 Expected results

The user is prompted for a comment but not for confirmation.
A new log file "wipemlog.txt" is created on the log disk.
The comment is logged.
The log file contains the correct information required by features 2.
The sectors of the media disk are correctly filled.

C1.6 Dkw-06

C1.6.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4, and especially whether diskwipe creates a log file with a name given on the command line in the *-log_name* option for a destination disk (instead of the default name wipedlog.txt).

C1.6.2 Test setup

Mount for example the SCS hard disk drive labeled "3B" in a slot on the test computer, for example "McMillan", such that it will be recognized as the Linux device /dev/sda. Reboot to Linux.

C1.6.3 Test case dependencies

None.

C1.6.4 Procedure

Run ***diskwipe*** using the options -noask and *-log_name*:

```
diskwipe dkw-06 mcmillan serban /dev/sda 3B -noask -
log_name dkwlog.txt
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Use ***diskedit*** or ***diskchg*** to examine the wiped hard disk's sectors.

C1.6.5 Expected results

diskwipe prompts the user for a comment but not for confirmation.
diskwipe creates a new log file "dkwlog.txt" on the log disk.
diskwipe logs the comment and the correct information required by feature 2.
diskwipe fills the sectors of the target disk correctly.

C1.7 Dkw-07

C1.7.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4, and especially whether diskwipe appends the log for a source disk to a log file with an alternate name when that file already exists.

C1.7.2 Test setup

Use the setup of *Dkw-06*.

C1.7.3 Test case dependencies

Dkw-06. Do not delete the log file with alternate name created in the case dkw-06.

C1.7.4 Procedure

Run ***diskwipe*** using the options -noask, *-src*, and *-log_name*. Use the same log file name as in dkw-06. Use a different fill character than in the case dkw-06.

```
diskwipe dkw-07 mcmillan serban /dev/sda 4B -noask -src -
log_name dkwlog.txt
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents.
Use ***diskedit*** or ***diskchg*** to examine the wiped hard disk's sectors.

C1.7.5 Expected results

diskwipe asks the user to confirm the alternate log file name, because he/she also specified the -*src* option.
diskwipe prompts the user for a comment but not for confirmation.
The log is appended to the old log file "dkwlog.txt", even though the log file was created for a destination disk and the user specified a source disk.
diskwipe logs the comment, the correct information required by feature 2, fills correctly the sectors of the target disk.

C1.8 Dkw-08

C1.8.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4, and especially whether diskwipe creates a new log file with a name given on the command line in the -log_name option, even though a log file with the same name exists and the -*new_log* option is used.

C1.8.2 Test setup

Use the setup of *Dkw-06*.

C1.8.3 Test case dependencies

Dkw-06 or dkw-07. Do not delete the log file created in those cases.

C1.8.4 Procedure

Run ***diskwipe*** using the options -noask, -*new_log*, and -*log_name* with the same log file name as in dkw-06 or dkw-07. Use a different fill character:

```
diskwipe dkw-08 mcmillan serban /dev/sda 5B -noask -new_log
-log_name dkwlog.txt
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents.
Use ***diskedit*** or ***diskchg*** to examine the wiped hard disk's sectors.

C1.8.5 Expected results

diskwipe prompts the user for a comment but not for confirmation.
diskwipe deletes the old log file "dkwlog.txt" and creates a new log file with the same name.
It logs the comment and the correct information required by feature 2.
diskwipe correctly fills the sectors of the target disk.

C1.9 Dkw-09

C1.9.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on a very large Serial ATA hard disk drive.

C1.9.2 Test setup

On the computer used for testing, mount a SATA hard disk drive, for example the one labeled "10B" (capacity 250GB). Reboot to Linux. On the machine we used ("frank"), the OS recognized the hard disk drive as device /dev/sda. We assume that the FS-TST 2.0 tools reside on the boot disk, which will also be used as log disk.

C1.9.3 Test case dependencies

None.

C1.9.4 Procedure

Run ***diskwipe*** with the mandatory arguments and with the option -*new_log*, -noask:

```
diskwipe dkw-09 frank serban /dev/sda AA -new_log -noask
```

Use the *ls* command and a text editor to examine the log file's existence, prescribed name, and contents.
Use ***diskedit*** or the -exam function of the ***diskchg*** tool to examine the sectors of the wiped hard disk.

C1.9.5 Expected results

diskwipe prompts the user for a comment. It creates a log file "wipedlog.txt", logs the command, the comment, the hard disk drive to be initialized, fills correctly the sectors, and logs the program execution.

C1.10 Dkw-10

C1.10.1 Purpose

The purpose of this test case is to test feature 2 when diskwipe is run with no arguments, with incorrect arguments, or with the -*h* option.

C1.10.2 Test setup

None.

C1.10.3 Test case dependencies

None.

C1.10.4 Procedure

Run ***diskwipe*** with no arguments and capture its output on the standard output into a file.
Run ***diskwipe*** with incorrect arguments and append its standard output to the same file.

Run ***diskwipe*** with the *-h* option alone on the command line and append its standard output to the same file.
Run ***diskwipe*** with the *-h* option together with other options on the command line and append its standard output to the same file.

```
diskwipe > output.txt
diskwipe dkw-10 mcmillan serban /dev/hdb 7F -logname >>
output.txt
diskwipe -h >> output.txt
diskwipe dkw-10 mcmillan serban /dev/sda CC -h >>
output.txt
```

Use a text editor to examine the contents of output.txt.

C1.10.5 Expected results

diskwipe displays its usage mode in each case.

C2 *partab* Test Case Specifications

C2.1 Ptb-01

C2.1.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 6 on a hard disk drive without partitions. We will use the -all option to log the empty entries of the partition table. We will use a one-word comment introduced by the *-comment* option on the command line.

C2.1.2 Test setup

We distinguish two cases: the partition table does not exist; and the partition table does exist but all entries are empty.
To set up the first case, reboot the test computer to Linux and zero the first sector of the test hard disk drive using the ***diskchg***.
To set up the second case, reboot the test computer to MS-DOS using the FS-TST1.0 boot diskette, then use PartitionMagic to create a partition, then delete all partitions on the test hard disk. Reboot to Linux.

C2.1.3 Test case dependencies

None.

C2.1.4 Procedure

For the first case - no partition table - run *partab* using the options -all and *-comment* followed by a one-word comment (assume that we selected a SCSI disk drive as target, identified as device /dev/sda):

```
partab ptb-01 mcmillan serban /dev/sda CC -all -comment NoTable
```

For the second case - a partition table with only empty entries - run *partab* again using the same log file to append the log records:

```
partab ptb-01 mcmillan serban /dev/sda CC -all -comment EmptyTable
```

Use the *ls* command and a text editor to examine the log file's existence and contents.
Use ***diskedit*** and/or Partition Magic to examine the partition table.

C2.1.5 Expected results

A (new) log file "pt-sda-log.txt" is created on the log disk.
The comment is logged.
The log file contains the correct information required by features 1 and 2.
If the partition table exists, 4 empty entries should be displayed. If the partition table does not exist, an error message should be logged.

C2.2 Ptb-02

C2.2.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, 5, and 6 for a disk with a primary FAT16 partition. We will test whether *partab* correctly logs the partition table entries as prescribed in the FS-TST specifications. We will also use the option -all to log the empty entries. We will test whether *partab* creates a new log file even though a log file with the same name exists, by using the option -*new_log*. We will also test whether a multi-word comment entered on the command line is correctly logged.

C2.2.2 Test setup

On the same target disk as in the case ptb-01 create a primary FAT16 partition using PartitionMagic (you need to first boot to MS-DOS). Reboot to Linux.

C2.2.3 Test dependencies

Ptb-01.

C2.2.4 Procedure

Run *partab* using the options -all, -*new_log*, and -*comment* followed by a multi-word comment:

```
partab ptb-02 mcmillan serban /dev/sda CC -new_log -all -
comment “Primary FAT16 partition”
```

Use the *ls* command and a text editor to examine the log file’s existence and contents.
Use ***diskedit*** and/or Partition Magic to examine the hard disk’s partition table(s).

C2.2.5 Expected results

A new log file “pt-sda-log.txt” is created on the log disk, even though a log file with the same name existed.
The comment is logged.
The log file contains the correct information required by features 1 and 2.
The partition table entries (feature 3) are correctly displayed. Each partition receives a unique ID number.
The empty partition table entries (feature 6) are also displayed.

C2.3 Ptb-03

C2.3.1 Purpose

The purpose of this test is to test features 1, 2, 3, 4, 5, and 6 on a disk with a primary FAT32 partition. Among other things, we will test whether *partab* appends the log records to the existing log file for the specified disk, and whether partab prompts the user to enter a comment.

C2.3.2 Test setup

Delete all existing partitions and create a primary FAT32 partition on the target disk used in Ptb-02, by using PartitionMagic (you need to boot first to MS-DOS). Reboot to Linux.

C2.3.3 Test dependencies

Ptb-02, in order to use the same log file to append the log records.

C2.3.4 Procedure

Run *partab*:

```
partab ptb-03 mcmillan serban /dev/sda CC -all
```

Use the *ls* command and a text editor to examine the log file's existence and contents.
Use ***diskedit*** and/or Partition Magic to examine the target disk's partition table(s).

C2.3.5 Expected results

partab appends the log to the log file named pt-sda-log.txt.
An interactively entered comment is correctly logged.
The target hard disk is correctly logged.
The log file contains the correct information required by feature 2.
The partition table entries (feature 3) are correctly displayed; each entry is given a unique id.
The empty partition table entries (feature 6) are also displayed.

C2.4 Ptb-04

C2.4.1 Purpose

The purpose of this test is to test features 1, 2, 3, 4, 5, and 6 on an IDE disk with a primary NTFS partition. Among other things, we will test whether partab creates a log file with the name specified on the command line using the option *-log_name*.
We will also test whether the default log file name depends on the target hard disk drive, by running again *partab* without the *-log_name* option.

C2.4.2 Test setup

Insert the target disk (for example the IDE disk externally labeled "7F") in a slot corresponding to the Linux device /dev/hdb. Create a primary NTFS partition using PartitionMagic (you need to boot first to MS-DOS for that). Reboot to Linux.

C2.4.3 Test dependencies

None.

C2.4.4 Procedure

Run *partab* using the options -all and *-log_name*:

```
partab ptb-04 mcmillan serban /dev/hdb 7F -all -log_name ptblog.txt
```

Then run *partab* again without the *-log_name* option:

```
partab ptb-04 mcmillan serban /dev/hdb 7F -all
```

Use the *ls* command and a text editor to examine the log files' existence and contents.
Use ***diskedit*** and/or Partition Magic to examine the hard disk's partition table(s).

C2.4.5 Expected results

First *partab* command creates a new log file with the name "ptblog.txt".
The user is prompted for a comment.
The disk, the comment, and all information required by feature 2 are correctly logged.
The partition table entries (feature 3) are correctly displayed and given unique ids.
The empty partition table entries (feature 6) are also displayed.

The second *partab* command creates a log file with the name "pt-hdb-log.txt", with a similar content.

C2.5 Ptb-05

C2.5.1 Purpose

The purpose of this test is to test features 1, 2, 3, 4, 5, and 6 on an IDE disk that contains a large (> 8GB) primary FAT32 partition, primary Linux Ext2 partition, and a Linux swap partition. We intend also to test log whether *partab* appends the log to an existing log file with the name specified in the -*log_name* option.

C2.5.2 Test setup

On the same target disk drive as in Ptb-04, create the large FAT32, Linux ext2, and Linux swap partitions using PartitionMagic (you need to boot first to MS-DOS for that). Reboot to Linux.

C2.5.3 Test dependencies

Ptb-04.

C2.5.4 Procedure

Run *partab* using the options -all and -log_name with the same alternate file name as in ptb-04:

```
partab ptb-05 mcmillan serban /dev/hdb 7F -all -log_name
ptblog.txt
```

Use the *ls* command and a text editor to examine the log file's existence and contents.
Use ***diskedit*** and/or Partition Magic to examine the hard disk's partition table.

C2.5.5 Expected results

partab appends the log to the old log file "ptblog.txt" created in the previous case.
The user is prompted for a comment.
The comment, the disk drive, and all information required by feature 2 are correctly logged.
The partition table entries (feature 3) are correctly displayed. Note that it is possible that the C/H/S start and end addresses of the partitions can be written incorrectly in the

partition table entries if they extend beyond cylinder 1024. *partab* should display the values it finds in the table.
The empty partition table entries (feature 6) are also displayed.

C2.6 Ptb-06

C2.6.1 Purpose

The purpose of this test is to test features 1, 2, 3, 4, 5, and 6 on a disk drive with a primary FAT16, primary FAT32 hidden, and a HPFS partition. Among other things, we will test whether a new log file with the name specified in the *-log_name* option is created when the option *-new_log* is used, even though the file existed from a previous case.

C2.6.2 Test setup

On the same target disk drive used in the case ptb-05, reboot to MS-DOS, and use PartitionMagic to delete all partitions and to create: a primary FAT16, a primary FAT32 hidden, a primary HPFS hidden, and a primary unformatted partition. Reboot to Linux.

C2.6.3 Test dependencies

Ptb-05.

C2.6.4 Procedure

Run *partab* using the options -all, *-new_log*, and log_name with the same alternate log file name as in ptb-05:

```
partab ptb-06 mcmillan serban /dev/hdb 7F -all -new_log -
log_name ptblog.txt
```

Use the *dir* command and a text editor to examine the log file's existence and contents.
Use ***diskedit*** and/or Partition Magic to examine the hard disk's partition table(s).

C2.6.5 Expected results

A new log file with the name "ptblog.txt" is created, even though a log file with the same name already exists.
The user is prompted for a comment.
The comment, the drive properties, and all information required by feature 2 are correctly logged.
The partition table entries (feature 3) are correctly displayed, with the type of hidden partitions specified as "unknown" or "other".
The empty partition table entries (feature 6) are also displayed.

C2.7 Ptb-07

C2.7.1 Purpose

The purpose of this test is to test features 1, 2, 3, 4, 5, and 6 on a SCSI disk drive with multiple extended and logical partitions. Among other things, we will test whether the name of the log file is unique for the target disk drive.

C2.7.2 Test setup

On a target disk drive mounted as device /dev/sda for example, create multiple primary and logical partitions, for example a primary FAT32, a primary Linux Ext2, and a primary extended partition which contains logical partitions FAT16, FAT32, and NTFS. Reboot to Linux.

C2.7.3 Test dependencies

None.

C2.7.4 Procedure

Run *partab* using the options -all and -*new_log*:

```
partab ptb-07 mcmillan serban /dev/sda CC -all -new_log
```

Use the *ls* command and a text editor to examine the log file's existence and contents.
Use ***diskedit*** and/or ***diskchg*** to examine the hard disk's partition table(s).

C2.7.5 Expected results

A new log file with the name "pt-sda-log.txt" is created.
The user is prompted for a comment.
The comment, the drive properties, and all information required by feature 2 are correctly logged.
The partition table entries (feature 3) are correctly displayed. The empty partition table entries (feature 6) are also displayed.

C2.8 Ptb-08

C2.8.1 Purpose

The purpose of this test is to test feature 2, namely how partab behaves when its command line contains no arguments, or contains incorrect arguments, or one of the options is -h.

C2.8.2 Test setup

Boot to Linux.

C2.8.3 Test dependencies

None.

C2.8.4 Procedure

Run *partab* with no arguments on the command line and capture its standard output to a file. Then run partab with incorrect arguments, with *-h* as the only argument, or with *-h* together with other arguments on the command line. Append the standard output to the same file.

```
partab > output.txt
partab ptb-08 mcmillan serban /dev/hdb 7F -logname >> output.txt
partab -h >> output.txt
partab ptb-08 mcmillan serban /dev/sda CC -h >> output.txt
```

C2.8.5 Expected results

partab displays its usage mode.

C3 *diskchg* Test Case Specifications

C3.1 dch-01

C3.1.1 Purpose

The purpose of this test case is to test ***diskchg***'s features 1, 2, 3, 8, and 9 on the target computer.

C3.1.2 Test setup

Select and insert a SCSI hard disk (for example, the one labeled "CC") into the drive that will be identified as the device /dev/sda. Boot to Linux; we assume that the Linux OS disk also contains the FS-TST tools executables. It will also be the log disk.

C3.1.3 Test case dependencies

None.

C3.1.4 Procedure

Delete all log files from the log disk.
Wipe out the target hard disk (device /dev/sda) using 0xCC as the fill value:

```
diskwipe dch-01 mcmillan serban /dev/sda CC
```

Run ***diskchg*** using a one-word comment, and the -exam option to start an interactive examination of the disk contents:

```
diskchg dch-01 mcmillan serban /dev/sda -exam -comment Examine
```

When prompted, enter the LBA addresses of the first sector (0), last sector (extract this information from diskwipe's log file; for example, in case of the disk "CC", this is 71687369), and other arbitrary sectors. Then enter the CHS addresses of the same sectors.

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Use ***diskedit*** to display the same sectors and compare results.

C3.1.5 Expected results

A (new) log file with the name characteristic for the hard disk drive and the -exam function ("cg-sda-xlog.txt") is created on the log disk.
The comment is logged.
The log file contains the correct information required by feature 2.
diskchg correctly converts the LBA addresses to C/H/S addresses.
All sectors are displayed correctly (use diskedit and diskwipe to assess correctness).

C3.2 dch-02

C3.2.1 Purpose

The purpose of this test case is to test ***diskchg***'s features 1, 2, 3, 8, and 9 on the target computer (especially: appending the log records to an existing log file, logging of a multi-word comment, and the correctness of the -exam function).

C3.2.2 Test setup

Use the setup of case *dch-01*.

C3.2.3 Test dependencies

dch-01, in order to have the log file for the same disk and function (-exam) present.

C3.2.4 Procedure

Run this test case after *dch-01*.
Run ***diskchg*** using a multi-word comment, and the -exam option to start an interactive examination of the disk contents:

```
diskchg dch-02 mcmillan serban /dev/sda -exam -comment
"Examining sectors, appending log"
```

When prompted, enter the CHS addresses of contiguous sectors from the end of a track and the beginning of the next track. Then enter the LBA addresses of the same sectors. Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Use ***diskedit*** to display same sectors and compare results.

C3.2.5 Expected results

The log records are appended to the log file created in the case *dch-01* (named cg-sda-xlog.txt).
The comment is logged.
The log file contains the correct information required by feature 2.
diskchg correctly converts the C/H/S addresses to LBA addresses.
The sectors are displayed correctly (use diskedit to assess correctness).

C3.3 dch-03

C3.3.1 Purpose

The purpose of this test case is to test ***diskchg***'s features 1, 2, 3, 8, and 9 on the target computer.

C3.3.2 Test setup

Use the setup of case *dch-01*.

C3.3.3 Test dependencies

dch-01 or *dch-02*, in order to have the log file for the same disk and function (-exam) present.

C3.3.4 Procedure

Run this test case after *dch-01* or *dch-02*.
Run ***diskchg*** using the -exam option to start an interactive examination of the disk contents, and the *-new_log* option to delete the previous log file for the -exam function. Do not use *-comment*, in order to test interactive comments:

```
diskchg dch-03 mcmillan serban /dev/sda -new_log -exam
```

When prompted, enter the LBA or CHS address of a sector outside the range of the test hard disk drive (for example, you may use the LBA address 71687370 or greater for the disk labeled "CC").
Use the *ls* command and a text editor to examine the log file's existence, name, and contents.

C3.3.5 Expected results

A new log file with the name cg-sda-xlog.txt is created, even though a log file with the same name already existed.
The user is prompted for a comment and the comment is logged.
The log file contains the correct information required by feature 2.
diskchg logs the fact that the sector address is invalid or displays a read error.

C3.4 dch-04

C3.4.1 Purpose

The purpose of this test case is to test ***diskchg***'s features 1, 2, 3, 7, and 9 on the target computer.

C3.4.2 Test setup

Use the setup of case *dch-01*.

C3.4.3 Test dependencies

None.

C3.4.4 Procedure

Run ***diskchg*** with the -read option to read and display a (portion of a) specified sector of the hard disk drive. Enter the sector address in the LBA format:

```
diskchg dch-04 mcmillan serban /dev/sda -read 80388 0 32
```

Use the *ls* command and a text editor to examine the log file's existence and contents.
Use ***diskedit*** to display the same sector.

C3.4.5 Expected results

The user is prompted for a comment.
A new log file cg-sda-rlog.txt is created.
The user is prompted for a comment, which is logged.
The log file contains the correct information required by feature 2.

diskchg correctly converts the LBA address to C/H/S format.
diskchg correctly displays the specified number of bytes starting at the specified offset within the specified sector. Use ***diskedit*** to assess correctness.

C3.5 dch-05

C3.5.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 7, and 9 on the target computer.

C3.5.2 Test setup

Use the setup of case *dch-01*.

C3.5.3 Test dependencies

None.

C3.5.4 Procedure

Run ***diskchg*** using the *-new_log* option and the *-read* option with the C/H/S address of the same sector as in the *dch-04* case, but with an offset too large for the capacity of a sector:

```
diskchg dch-05 mcmillan serban /dev/sda -new_log -read 5/1/1 640 32
```

Use the *ls* command and a text editor to examine the log file's existence and contents.
Use ***diskedit*** to display the same sector.

C3.5.5 Expected results

diskchg created a new log file cg-sda-rlog.txt
diskchg prompts the user for a comment, which is logged.
diskchg logs the correct information required by feature 2.
diskchg resets the offset to zero, then correctly displays the specified number of bytes starting at the reset offset within the specified sector. Use diskedit and diskwipe documentation to assess correctness.

C3.6 dch-06

C3.6.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 7, and 9 on the target computer.

C3.6.2 Test setup

Use the setup of case *dch-01*.

C3.6.3 Test dependencies

None.

C3.6.4 Procedure

Run ***diskchg*** using the *-new_log* option and the -read option with a length too large for the capacity of a sector, to read and display a (portion of a) specified sector of the hard disk drive:

```
diskchg dch-06 mcmillan serban /dev/sda -read 5/1/1 0 1024
-new_log
```

Use the *ls* command and a text editor to examine the log file's existence and contents.
Use ***diskedit*** to display the same sector.

C3.6.5 Expected results

diskchg creates a new log file with a name characteristic for the -read function ("cg-sda-rlog.txt") is created. diskchg prompts the user for a comment, which is logged.
diskchg logs the correct information required by feature 2.
diskchg resets the length to 16, then correctly displays min(16, 512-offset) bytes starting at the specified offset within the specified sector. Use diskedit and diskwipe documentation to assess correctness.

C3.7 dch-07

C3.7.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 7, and 9 on the target computer.

C3.7.2 Test setup

Use the setup of case *dch-01*.

C3.7.3 Test dependencies

None.

C3.7.4 Procedure

Run ***diskchg*** using the *-new_log* option and the -read option with valid offset and length values, but with offset+count too large, to read and display a (portion of a) specified sector of the hard disk drive:

```
diskchg dch-07 mcmillan serban /dev/sda -read 5/1/1 256 400
-new_log
```

Use the *ls* command and a text editor to examine the log file's existence and contents.
Use ***diskedit*** to display the same sector.

C3.7.5 Expected results

diskchg creates a new log file with a name characteristic for the -read function ("cg-sda-rlog.txt"). diskchg prompts the user for a comment, which is logged.
diskchg logs the correct information required by feature 2.

diskchg correctly displays only the bytes starting at the specified offset up to the end of the specified sector. Use diskedit and diskwipe documentation to assess correctness.

C3.8 dch-08

C3.8.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 7, and 9 on the target computer.

C3.8.2 Test setup

Use the setup of case *dch-01*.

C3.8.3 Test dependencies

None.

C3.8.4 Procedure

Run ***diskchg*** using the *-new_log* option and the -read option with an invalid sector address (for example, 71687370 or greater for the disk labeled "CC"):

```
diskchg dch-08 mcmillan serban /dev/sda -new_log -read
71687370 0 512
```

Use the *ls* command and a text editor to examine the log file's existence and contents.

C3.8.5 Expected results

diskchg creates a new log file with a name characteristic for the -read function ("cg-sda-rlog.txt"). diskchg prompts the user for a comment, which is logged.
diskchg logs the correct information required by feature 2.
diskchg logs the fact that the sector address is invalid or displays a read error.

C3.9 dch-09

C3.9.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 5, 9 on the target computer. Mainly, we intend to test whether diskchg correctly fills a sector when using the -fill option with the detected disk geometry expressed through the value 0 for the heads argument.

C3.9.2 Test setup

Use the setup of case *dch-01*.

C3.9.3 Test dependencies

None.

C3.9.4 Procedure

Run ***diskchg*** three times:

(1) using the -*new_log* option and the -read option to read and display the initial content of a sector specified by its C/H/S address (called later the destination sector):

```
diskchg dch-09 mcmillan serban /dev/sda -new_log -read 5/1/1 0 32
```

(2) using the -*new_log* and -fill options to fill the destination sector as the source sector in the geometry reported by BIOS (use 0 as the value of the parameter heads) and with a different fill value:

```
diskchg dch-09 mcmillan serban /dev/sda -new_log -fill 5/1/1 6/1/1 0 BB
```

(3) using the -read option to read the destination sector after filling:

```
diskchg dch-09 mcmillan serban /dev/sda -read 5/1/1 0 32
```

Use the *ls* command and a text editor to examine the log files' names and contents.
Use ***diskedit*** to display the specified sectors and compare the reported values with the logged ones.

C3.9.5 Expected results

diskchg creates a new log file with a name characteristic for the -read function ("cg-sda-rlog.txt") when run the first time; the second and fourth times, diskchg appends the log to the existing log file. diskchg creates a new log file with a name characteristic for the -fill function ("cg-sda-flog.txt") when run the third time.
diskchg prompts the user for comments, which are logged.
diskchg logs the correct information required by feature 2.
diskchg correctly converts the C/H/S addresses to LBA.
diskchg correctly fills and displays the specified sector. Use diskedit and diskwipe documentation to assess correctness.

C3.10 dch-10

C3.10.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 5, 9 on the target computer. It is very similar to the case dch-09; the only difference is that we instruct diskchg to use the detected geometry by specifying that geometry explicitly in the heads parameter, instead of letting diskchg discover it.

C3.10.2 Test setup

Use the setup of case *dch-01*.

C3.10.3 Test dependencies

None.

C3.10.4 Procedure

Run ***diskchg*** four times:

(1) using the *-new_log* option and the -read option to read and display the initial content of a sector specified by its C/H/S address (called later the destination sector):

```
diskchg dch-10 mcmillan serban /dev/sda -new_log -read 5/1/1 0 32
```

(2) using the *-new_log* and -fill options to fill the destination sector as the source sector in the geometry reported by BIOS given explicitly on the command line as the value of the parameter heads and a different fill value (in the following command we assumed that the geometry detected is 255 heads/cyl):

```
diskchg dch-10 mcmillan serban /dev/sda -new_log -fill 5/1/1 6/1/1 255 AA
```

(3) using the -read option to read the destination sector after filling:

```
diskchg dch-10 mcmillan serban /dev/sda -read 5/1/1 0 32
```

Use the *ls* command and a text editor to examine the log files' names and contents.
Use ***diskedit*** to display the specified sectors and compare the reported values with the logged ones.

C3.10.5 Expected results

diskchg creates a new log file with a name characteristic for the -read function ("cg-sda-rlog.txt") when run the first time; the second and fourth times, diskchg appends the log to the existing log file. diskchg creates a new log file with a name characteristic for the -fill function ("cg-sda-flog.txt") when run the third time.
diskchg prompts the user for comments, which are logged.
diskchg logs the correct information required by feature 2.
diskchg correctly converts the C/H/S addresses to LBA.
diskchg correctly fills and displays the specified sector. Use diskedit and diskwipe documentation to assess correctness.

C3.11 dch-11

C3.11.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 5, 9 on the target computer. Mainly, we intend to test whether diskchg correctly fills a sector when using the -fill option with a disk geometry different from the one detected.

C3.11.2 Test setup

Use the setup of the case dch-01.

C3.11.3 Test dependencies

None.

C3.11.4 Procedure

Run ***diskchg*** three times:

(1) using the -read option to read and display the initial contents of the target/destination sector of the hard disk drive:

```
diskchg dch-11 mcmillan serban /dev/sda -new_log -read 5/1/1 0 32
```

(2) using -*new_log* and the -fill option to fill the target sector with the contents of the source sector in a geometry different from that detected:

```
diskchg dch-11 mcmillan serban /dev/sda -new_log -fill 5/1/1 6/1/1 200 DD
```

(3) using the -read option to read the destination sector after filling:

```
diskchg dch-11 mcmillan serban /dev/sda -read 5/1/1 0 32
```

Use the *ls* command and a text editor to examine the log files names and contents.
Use ***diskedit*** to display the specified sectors and compare the reported values with the logged ones. Compare the contents of the target and source sectors.

C3.11.5 Expected results

diskchg creates a new log file with a name characteristic for the -read function (“cg-sda-rlog.txt”) when run the first time; the second and fourth times, diskchg appends the log to the existing log file. diskchg creates a new log file with a name characteristic for the -fill function (“cg-sda-flog.txt”) when run the third time.
diskchg prompts the user for comments, which are logged.
diskchg logs the correct information required by feature 2.
diskchg correctly converts the C/H/S addresses to LBA.
diskchg correctly fills and displays the specified sector. Use diskedit and diskwipe documentation to assess correctness. The first 26 bytes (containing the sector’s CHS and LBA address) of the source and target sectors should be identical. The rest of the target sector should be filled with 0xDD.

C3.12 dch-12

C3.12.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 6, and 9 on the target computer, when we instruct diskchg to change the value of a certain byte of a sector specified by its LBA address by using the -write option.

C3.12.2 Test setup

Use the setup of the case *dch-01*.

C3.12.3 Test dependencies

None.

C3.12.4 Procedure

Run ***diskchg*** three times:

(1) using *-new_log* and the -read option to read and display the initial contents of a specified sector of the hard disk drive:

```
diskchg dch-12 mcmillan serban /dev/sda -new_log -read 80388 0 32
```

(2) using *-new_log* and the -write option with a LBA address to set a specified byte of that sector to a specified value, e.g., 0xCE:

```
diskchg dch-12 mcmillan serban /dev/sda -new_log -write 80388 26 CE
```

(3) using the -read option to read the specified sector after setting:

```
diskchg dch-12 mcmillan serban /dev/sda -read 80388 0 32
```

Use the *ls* command and a text editor to examine the log files names and contents.
Use ***diskedit*** to display the specified sector and compare the reported value with the logged one.

C3.12.5 Expected results

diskchg creates a new log file with a name characteristic for the invoked ***diskchg*** function each time the *-new_log* option is used; otherwise, diskchg appends the log to the existing log file.
diskchg prompts the user for a comment if the *-comment* option is not used and logs the comment.
diskchg logs the correct information required by feature 2.
diskchg correctly translates the LBA address to C/H/S.
The specified byte (in our example the byte at offset 26 within the sector 5/1/1) has the value 0xCE. The rest of that sector is unchanged.

C3.13 dch-13

C3.13.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 6, and 9 on the target computer, when we instruct diskchg to change the value of a certain byte of a sector specified by its C/H/S address when using the -write option.

C3.13.2 Test setup

Use the setup of the case *dch-01*.

C3.13.3 Test dependencies

None.

C3.13.4 Procedure

Run ***diskchg*** three times:

(1) using *-new_log* and the -read option to read and display the initial contents of a specified sector of the hard disk drive:

```
diskchg dch-13 mcmillan serban /dev/sda -new_log -read 5/1/1 0 32
```

(2) using *-new_log* and the -write option with a C/H/S address to set a specified byte of that sector to a specified value, e.g., 0xCE:

```
diskchg dch-13 mcmillan serban /dev/sda -new_log -write 5/1/1 26 CE
```

(3) using the -read option to read the specified sector after setting:

```
diskchg dch-13 mcmillan serban /dev/sda -read 5/1/1 0 32
```

Use the *ls* command and a text editor to examine the log files names and contents.
Use ***diskedit*** to display the specified sector and compare the reported value with the logged one.

C3.13.5 Expected results

diskchg creates a new log file with a name characteristic for the invoked ***diskchg*** function each time the *-new_log* option is used; otherwise, diskchg appends the log to the existing log file.
diskchg prompts the user for a comment if the *-comment* option is not used and logs the comment.
diskchg logs the correct information required by feature 2.
The specified byte (in our example the byte at offset 26 within the sector 5/1/1) has the value 0xCE. The rest of that sector is unchanged.

C3.14 dch-14

C3.14.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 6, and 9 on the target computer, when we instruct diskchg to change the value of a certain byte of a sector specified by its C/H/S address when using the -write option with an invalid byte offset.

C3.14.2 Test setup

Use the setup of the case *dch-01*.

C3.14.3 Test dependencies

None.

C3.14.4 Procedure

Run ***diskchg*** using the *-new_log* and the -write option with a C/H/S or LBA address, and a byte offset too large for the sector capacity:

```
diskchg dch-14 mcmillan serban /dev/sda -new_log -write 5/1/1 640 CF
```

Use the *ls* command and a text editor to examine the log file name and contents.

C3.14.5 Expected results

diskchg creates a new log file cg-sda-wlog.txt.
diskchg prompts the user for a comment and logs the comment.
diskchg logs the fact that the byte offset is too large and exits.

C3.15 dch-15

C3.15.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 6, and 9 on the target computer, when we instruct diskchg to change the value of a certain byte of a sector whose address is invalid, i.e., outside the disk range.

C3.15.2 Test setup

Use the setup of the case *dch-01*.

C3.15.3 Test dependencies

None.

C3.15.4 Procedure

Run ***diskchg*** using the *-new_log* and the -write option with a C/H/S or LBA address that specifies a sector outside the hard disk range:

```
diskchg dch-15 mcmillan serban /dev/sda -new_log -write 71687370 26 DD
```

Use the *ls* command and a text editor to examine the log file name and contents.

C3.15.5 Expected results

diskchg creates a new log file cg-sda-wlog.txt.
diskchg prompts the user for a comment and logs the comment.
diskchg logs the fact that the sector address is outside the valid range, or reports a read error and terminates.

C3.16 dch-16

C3.16.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 9 on the target computer, when we zero the first sector of the disk by using diskchg's -zero option with the LBA address of the sector.

C3.16.2 Test setup

Mount a IDE disk, for example the one labeled "7F" in a slot of the test computer, such that the corresponding Linux device is /dev/hdb. Reboot to Linux.

C3.16.3 Test dependencies

None.

C3.16.4 Procedure

Run ***diskchg*** twice:

(1) using the -zero option to set to zero the first sector of the target hard disk drive:

```
diskchg dch-16 mcmillan serban /dev/hdb -zero 0
```

(2) using *-new_log* and the -read option to read and display the zeroed sector:

```
diskchg dch-16 mcmillan serban /dev/hdb -new_log -read 0 0
128
```

Use the *ls* command and a text editor to examine the log files names and contents.
Use ***diskedit*** to display the specified sector and compare the reported value with the logged one.

C3.16.5 Expected results

diskchg creates a log file with the name characteristic for the invoked ***diskchg*** function and the device name each time the *-new_log* option is used or if a log file does not exist; otherwise, the log is appended to the existing log file.
diskchg prompts the user for a comment, which is logged.
diskchg logs the correct information required by feature 2.
diskchg correctly translates the LBA address to C/H/S.
diskchg correctly fills the specified sector with zeroes.

C3.17 dch-17

C3.17.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 9 on the target computer, when we zero the last sector of the disk by using diskchg's -zero option with the C/H/S address of the sector and creating a log file with the name specified in the *-log_name* option.

C3.17.2 Test setup

Same as in *dch-16*.

C3.17.3 Test dependencies

None.

C3.17.4 Procedure

Run ***diskchg*** twice:

(1) using the -zero option to set to zero the last sector of the target hard disk drive, and the -*log_name* option to set the log file name (in the command below we assumed that the C/H/S address of the last sector is 4866/87/21; you may extract this information from the value written by ***diskwipe*** in the last sector of the disk):

```
diskchg dch-17 mcmillan serban /dev/hdb -new_log -log_name
zerolog.txt -zero 4866/87/21
```

(2) using the -read option to read and display the zeroed sector. Also, use the -*log_name* option with the same name so that the results are appended to the same log file:

```
diskchg dch-17 mcmillan serban /dev/hdb -log_name
zerolog.txt -read 4462/84/48 0 128
```

Use the *ls* command and a text editor to examine the log files names and contents.
Use ***diskedit*** to display the specified sector and compare the reported value with the logged one.

C3.17.5 Expected results

diskchg creates a log file “zerolog.txt”.
diskchg prompts the user for a comment, which is logged.
diskchg logs the correct information required by feature 2.
diskchg correctly fills the specified sector with zeroes.
The second run of ***diskchg*** appends the log record to the zerolog.txt file.

C3.18 dch-18

C3.18.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 9 on the target computer, when we zero an arbitrary sector of the disk by using diskchg’s -zero option with the LBA address of the sector. We also test whether the -*new_log* and -*log_name* options used together trigger the creation of a new log file of specified name when a log file with the same name already exists.

C3.18.2 Test setup

Same as in *dch-16*.

C3.18.3 Test dependencies

dch-17, in order for the log file “zerolog.txt” to exist.

C3.18.4 Procedure

Run ***diskchg*** twice:

(1) using the *-zero* option to set to zero a sector of the target hard disk drive, and the *-new_log* and *-log_name* option to create a new log file with a specified name, even though a log file with the same name was created in the previous case:

```
diskchg dch-18 mcmillan serban /dev/hdb -new_log -log_name
zerolog.txt -zero 80388
```

(2) using the *-log_name* with the same log file name as in the first run, and the -read option to read and display the zeroed sector:

```
diskchg dch-18 mcmillan serban /dev/hdb -log_name
zerolog.txt -read 80388 0 128
```

Use the *ls* command and a text editor to examine the log files names and contents.
Use ***diskedit*** to display the specified sector and compare the reported value with the logged one.

C3.18.5 Expected results

diskchg creates a new log file “zerolog.txt”, even though it already existed.
diskchg prompts the user for a comment, which is logged.
diskchg logs the correct information required by feature 2.
diskchg correctly fills the specified sector with zeroes.

C3.19 dch-19

C3.19.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 9 on the target computer, when we try to zero a sector with an invalid LBA address.

C3.19.2 Test setup

Same as in *dch-16*.

C3.19.3 Test dependencies

None.

C3.19.4 Procedure

Run ***diskchg*** using the -zero option to set to zero a sector with an LBA address outside the valid range of sectors for the target hard disk:

```
diskchg dch-19 mcmillan serban /dev/hdb -new_log -zero
78177792
```

Use the *ls* command and a text editor to examine the log files names and contents.
Use ***diskedit*** to display the specified sector and compare the reported value with the logged one.

C3.19.5 Expected results

diskchg creates a log file “zerolog.txt”.
diskchg prompts the user for a comment, which is logged.
diskchg logs the correct information required by feature 2.
diskchg reports that the sector address is invalid or a write error.

C3.20 dch-20

C3.20.1 Purpose

The purpose of this test case is to test ***diskchg***’s features 1, 2, 3, 7, and 9 on a Serial ATA hard disk drive of large capacity.

C3.20.2 Test setup

Mount a SATA disk (for example the one labeled “10B”) on the test computer (for example “frank”). We assume the corresponding Linux device is /dev/sdb. Reboot to Fedora Core 3, which runs on “frank”.

C3.20.3 Test dependencies

None.

C3.20.4 Procedure

Run ***diskchg*** three times with the -read option to read and display the first and last sectors, and to try to read a sector beyond the disk range:

```
diskchg dch-20 frank serban /dev/sdb -read 0 0 32
diskchg dch-20 frank serban /dev/sdb -read 488397167 0 32
diskchg dch-20 frank serban /dev/sdb -read 488397168 0 32
```

Use the *ls* command and a text editor to examine the log file’s existence and contents.
Use ***diskedit*** to display the sectors, or trust ***diskwipe***, if, as in our case, it was used to initialize the disk prior to running ***diskchg***.

C3.20.5 Expected results

diskchg prompts the user for comments.
A new log file cg-sdb-rlog.txt is created. The second and third commands append the records to the same log file.
diskchg logs the correct information required by feature 2.
diskchg correctly converts the LBA address to C/H/S format.
diskchg correctly displays the specified number of bytes starting at the specified offset within the specified sectors in the case of the first and second command. ***diskchg*** detects and displays the attempt to read a sector outside the correct disk range.

C3.21 dch-21

C3.21.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 6, and 9 on the target computer and a Serial ATA hard disk drive of large capacity.

C3.21.2 Test setup

Use the setup of the case *dch-20*.

C3.21.3 Test dependencies

None.

C3.21.4 Procedure

Run ***diskchg*** with the -write option twice, in order to modify a byte of the first and last sectors of the disk. Use the same log file:

```
diskchg dch-21 frank serban /dev/sdb -write 0 30 BB
diskchg dch-21 frank serban /dev/sdb -write 488397167 30 BB
```

Run ***diskchg*** twice with the -read option, in order to read the modified sectors. Use the same log file:

```
diskchg dch-21 frank serban /dev/sdb -read 0 0 32 -new_log
diskchg dch-21 frank serban /dev/sdb -read 488397167 0 32
```

We can assess the correctness, because the disk was initialized using ***diskwipe*** prior to this test.

C3.21.5 Expected results

diskchg creates a new log file with a name characteristic for the invoked ***diskchg*** function and Linux device each time the -new_log option is used; otherwise, diskchg appends the log to the existing log file.
diskchg prompts the user for comments if the *-comment* option is not used and logs the comment.
diskchg logs the correct information required by feature 2.
diskchg correctly translates the LBA address to C/H/S.
The specified byte (in our example the byte at offset 30) is set to 0xBB in both sectors. The rest of the sectors are unchanged.

C3.22 dch-22

C3.22.1 Purpose

The purpose of this test case is to test feature 2, namely whether ***diskchg*** displays its usage mode when invoked with *-h* as one of its arguments.

C3.22.2 Test setup

None.

C3.22.3 Test dependencies

None.

C3.22.4 Procedure

Run ***diskchg*** without arguments; using incorrect arguments; using the *-h* option alone on the command line; and using the option *-h* together with other options on the command line. Capture the standard output into a file:

```
diskchg > output.txt
diskchg dch-20 mcmillan serban /dev/sda -readlog 5/1/1 0 32
>> output.txt
diskchg -h >> output.txt
diskchg dch-20 mcmillan serban /dev/sda -h -read 5/1/1 0 32
>> output.txt
```

Examine the information displayed by ***diskchg*** on the standard output.

C3.22.5 Expected results

diskchg displays its usage information on the standard output.

C4 *seccmp* Test Case Specifications

C4.1 scm-01

C4.1.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and the selected hard disk drives. Specifically, we want to test whether ***seccmp*** creates a new log file with the default name when no log file is present, correctly logs the hard disk drives, a one-word comment entered on the command line, and the program execution. Also, we want to test whether ***seccmp*** compares the sectors and logs the result according to feature 4, when the sectors are *not* diskwipe-style- or zero-filled.

C4.1.2 Test setup

Select and insert a "source" and a "destination" disk (e.g., the disks labeled "CC", respectively "7F") in the target computers' drives. For the example disks, we assume that the source disk "CC" will be the Linux device /dev/sda, and the destination disk "7F" will be the Linux device /dev/hdb.
Boot up to Linux.

C4.1.3 Test case dependencies

None.

C4.1.4 Procedure

Delete all log files from the log disk (the Linux boot disk).

Use the -fill function of ***diskchg*** to fill first sector of each disk in diskwipe-style with 0xCC, 0x7F respectively. Then use the -write function of ***diskchg*** to modify the first sector of each disk, so that the sector is *not* diskwipe-style-filled or zero-filled. For example:

```
diskchg scm-01 mcmillan serban /dev/sda -fill 0 0 0 CC
diskchg scm-01 mcmillan serban /dev/sda -write 0 30 01
diskchg scm-01 mcmillan serban /dev/hdb -fill 0 0 0 7F
diskchg scm-01 mcmillan serban /dev/hdb -write 0 30 01
```

Run ***seccmp*** using a one-word comment and the -sector option specifying the first sector of each disk:

```
seccmp scm-01 mcmillan serban /dev/sda CC /dev/hdb 7F -
sector 0 0 -comment CompareNonFilledSectors
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Use ***diskedit*** to display the source and destination sectors and compare results.

C4.1.5 Expected results

seccmp creates a log file with the name "seclog.txt" on the log disk.

seccmp logs the comment and the correct information required by features 1, 2, 3.
seccmp correctly compares the sectors and logs all differences as required by feature 4.

C4.2 scm-02

C4.2.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 5 on the target computer and selected hard disk drives. Specifically, we want to test whether ***seccmp*** appends the log records to an existing log file, and correctly logs the hard disk drives, a multi-word comment entered on the command line, and the program execution. Also, we want to test whether seccmp compares diskwipe-style filled sectors and records the result according to requirements.

C4.2.2 Test setup

Use the setup of test case scm-01. Do not delete the log file created by test case scm-01.

C4.2.3 Test case dependencies

scm-01.

C4.2.4 Procedure

Run this test case right after scm-01. Do no delete the log file. Fill the last sector of each disk in diskwipe-style with 0xCC, 0x7F respectively. You may use the -fill function of ***diskchg*** for that, like this:

```
diskchg scm-02 mcmillan serban /dev/sda -new_log -fill
71687369 71687369 0 CC
diskchg scm-02 mcmillan serban /dev/hdb -new_log -fill
78177791 78177791 0 CC
```

Run ***seccmp*** using the *-comment* option with a multi-word comment, and the -sector option specifying the last sector of each disk:

```
seccmp scm-02 mcmillan serban /dev/sda CC /dev/hdb 7F -
sector 71687369 78177791 -comment “compare last sectors
diskwipe-filled”
```

Use the *ls* command and a text editor to examine the log file’s existence, name, and contents. Use ***diskedit*** to display the source and destination sectors and compare results.

C4.2.5 Expected results

seccmp appends the log records to the log file “seclog.txt” created by the preceding test case.
seccmp logs the comment and the correct information required by features 1, 2, 3.
seccmp determines that the sectors are diskwipe-style filled and different, and logs the number of different bytes, as required by feature 5.

C4.3 scm-03

C4.3.1 Purpose

The purpose of this test case is to test features 1, 2, 3 on the target computer and the selected hard disk drives. Specifically, we want to test whether ***seccmp*** creates a new log file when instructed to do so by the *-new_log* option, correctly logs the hard disk drives, prompts the user for a comment, and correctly logs the comment and the program execution. We also test whether ***seccmp*** correctly detects sectors out of disk range.

C4.3.2 Test setup

Use the setup of test case scm-02. Do not delete the log file created by test case scm-02.

C4.3.3 Test case dependencies

scm-02.

C4.3.4 Procedure

Run this test case right after scm-02. Do no delete the log file created in that test case.

Run ***seccmp*** without the *-comment* option, with *-new_log*, and the -sector option specifying sectors out of range for both the source and destination disks:

```
seccmp scm-03 mcmillan serban /dev/sda CC /dev/hdb 7F -
sector 71687370 78177792 -new_log
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents.

C4.3.5 Expected results

seccmp creates a new log file of name "seclog.txt" even though a file with the same name already exists.
seccmp prompts the user for a comment, which is logged.
seccmp logs the correct information required by features 1, 2, 3.
seccmp determines that the sector numbers are beyond the disk range and issues/logs an error message.

C4.4 scm-04

C4.4.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and the selected hard disk drives. Specifically, we want to test whether ***seccmp*** prompts the user for sector numbers and comment, compares and logs the correct result when both source and destination sectors are diskwipe-style filled. We will use the same fill value on the command line for both source and destination, over various combinations of actual source and destination fill values. When the actual source and destination fill values are identical, we will consider source and destination sectors with the same or different header (i.e., LBA address).

C4.4.2 Test setup

Use the hard disk drives labeled “CC” and “7F” as source and destination drive respectively. Assume that the two disks have the same number of heads per track. Using ***diskchg***, fill the following sectors of the source disk in diskwipe-style with the specified value:

Source sector (LBA)	Actual source fill value
1000	CC
1001	CD

Using ***diskchg***, fill the following sectors of the destination disk with the specified value:

Destination sector (LBA)	Actual destination fill value
1000	CC
1001	CD
1002	CE
2000	CC
2001	CD

C4.4.3 Test case dependencies

None.

C4.4.4 Procedure

Run ***seccmp*** using the *-new_log* option, but not the *-comment* or -sector options. Use the CC fill value on the command line for both source and destination:

```
seccmp scm-04 mcmillan serban /dev/sda CC /dev/hdb CC -new_log
```

When prompted for sector LBA addresses, enter the following source and destination sector addresses:

Source sector	Destination sector	Notes
1000	1000	Same actual source and destination fill values, equal to the specified fill value (on the command line), same sector headers.
1000	1001	Actual src = specified fill value, actual src != actual dst.
1001	1000	Actual src != specified fill value, actual src != actual dst.
1001	1002	Actual src != actual dst, actual src != specified fill value, actual dst !=specified fill value.
1001	1001	Actual src = actual dst, actual src != specified fill value, same sector headers.
1000	2000	Actual src = actual dst, actual src = specified fill

		value, different sector headers
1001	2001	Actual src = actual dst, actual src != specified fill value, different sector headers.

Use the *ls* command and a text editor to examine the log file's existence, name, and contents.

C4.4.5 Expected results

seccmp creates a new log file of name "seclog.txt".
seccmp prompts the user for a comment, and logs the comment.
seccmp prompts the user for sector addresses.
seccmp logs the correct information required by features 1, 2, 3.
seccmp compares the specified sectors and logs the correct result, as required by features 4 and 5.

C4.5 scm-05

C4.5.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and the selected hard disk drives. Specifically, we want to test whether ***seccmp*** prompts the user for sector numbers and comment, compares and logs the correct result when both source and destination sectors are diskwipe-style filled. We will use different source and destination fill values on the command line, over various combinations of actual source and destination fill values. Whenever the actual source and destination fill values are identical, we will consider source and destination sectors with the same or different header (i.e., LBA and C/H/S addresses).

C4.5.2 Test setup

Use the hard disk drives labeled "CC" and "7F" as source and destination drive respectively. Assume that the two disks have the same number of heads per track. Using ***diskchg***, fill the following sectors of the source disk in diskwipe-style with the specified value:

Source sector (LBA)	Actual source fill value
1000	CC
1001	CD
1002	7F

Using ***diskchg***, fill the following sectors of the destination disk with the specified value:

Destination sector (LBA)	Actual destination fill value
1000	CC
1001	CD
1002	7F
1003	7E
2000	CC

2001	CD
2002	7F

C4.5.3 Test case dependencies

None

C4.5.4 Procedure

Run ***seccmp*** using the *-new_log* option, but not the *-comment* or -sector options. Use CC as the source fill value and 7F as the destination fill value on the command line:

```
seccmp scm-05 mcmillan serban /dev/sda CC /dev/hdb 7F -
new_log
```

When prompted for sector LBA addresses, enter the following source and destination sector addresses:

Source sector	Destination sector	Notes
1000	1000	Actual src = spec. src, actual dst != spec dst, actual src = actual dst, same sector headers.
1000	1002	Actual src = spec src, actual dst = spec dst.
1000	1003	Actual src = spec src, actual src != actual dst.
1001	1001	Actual src != spec src, actual dst != spec dst, actual src = actual dst, same sector headers.
1001	1002	Actual src != spec src, actual dst = spec dst, actual src != actual dst.
1001	1003	Actual src != spec src, actual dst != spec dst, actual src != actual dst.
1002	1002	Actual src != spec src, actual dst = spec dst, actual src = actual dst, same sector headers.
1000	2000	Actual src = spec src, actual dst != spec dst, actual src = actual dst, different sector headers
1001	2001	Actual src != spec src, actual dst != spec dst, actual src = actual dst, different sector headers.
1002	2002	Actual src != spec src, actual dst = spec dst, actual src = actual dst, different sector headers.

Use the *ls* command and a text editor to examine the log file's existence, name, and contents.

C4.5.5 Expected results

seccmp creates a new log file of name "seclog.txt".
seccmp prompts the user for a comment, and logs the comment.
seccmp prompts the user for sector addresses.
seccmp logs the correct information required by features 1, 2, 3.

seccmp compares the specified sectors and logs the correct result, as required by features 4 and 5.

C4.6 scm-06

C4.6.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, 5, and 6 on the target computer and the selected hard disk drives. Specifically, we want to test whether ***seccmp*** creates a log file with a name specified in the -*log_name* option, prompts the user for sector numbers and comment, and correctly compares sectors that have diskwipe-style filled, zero-filled, or arbitrary contents. Note that both source and destination sectors with diskwipe-style fill were covered in the previous two cases.

C4.6.2 Test setup

Use the hard disk drives labeled "CC" and "7F" as source and destination drive respectively. Using ***diskchg***'s functions -fill, -zero, and -write, prepare some sectors of the source disk as specified in the following table:

Source sector (LBA)	Value
1000	diskwipe-style filled with CC
1001	Zero
1002	Neither zero nor diskwipe-style

Using ***diskchg***'s functions -fill, -zero, and -write, prepare some sectors of the source disk as specified in the following table:

Destination sector (LBA)	Actual destination fill value
2000	diskwipe-style filled with 7F
2001	Zero
2002	Neither zero nor diskwipe-style

C4.6.3 Test case dependencies

None

C4.6.4 Procedure

Run ***seccmp*** using the -*log_name* option, but not the -*comment* or -sector options. Use CC as the source fill value and 7F as the destination fill value on the command line:

```
seccmp scm-06 mcmillan serban /dev/sda CC /dev/hdb 7F -
log_name log.txt
```

When prompted for sector LBA addresses, enter the following source and destination sector addresses:

Source sector	Destination sector	Notes
1000	2001	src diskwipe-style, dst zero.
1000	2002	src diskwipe-style, dst not diskwipe-style, not zero.
1001	2000	src zero, dst diskwipe-style.
1001	2001	Src zero, dst zero.
1001	2002	Src zero, dst not diskwipe-style, not zero.
1002	2000	Src not diskwipe style, not zero, dst diskwipe-style.
1002	2001	Src not diskwipe style, not zero, dst zero.
1002	2002	Src not diskwipe style, not zero, dst not diskwipe-style, not zero.

Use the *ls* command and a text editor to examine the log file's existence, name, and contents.

C4.6.5 Expected results

seccmp creates a new log file of name "log.txt".
seccmp prompts the user for a comment, and logs the comment.
seccmp prompts the user for sector addresses.
seccmp logs the correct information required by features 1, 2, 3.
seccmp compares the specified sectors and logs the correct result, as required by features 4 and 5.

C4.7 scm-07

C4.7.1 Purpose

The purpose of this test case is to test feature 3 of the ***seccmp*** tool. Specifically, we want to test whether ***seccmp*** displays its usage mode when the seccmp command is invoked with the *-h* option.

C4.7.2 Test setup

None.

C4.7.3 Test case dependencies

None.

C4.7.4 Procedure

Run ***seccmp*** in the following cases: without any argument, with incorrect arguments, with the *-h* option single on the command line, and with the *-h* option together with correct arguments on the command line, and redirect the standard output to a file:

```
seccmp > output.txt
seccmp scm-07 mcmillan serban /dev/sda CC -logname >> output.txt
```

```
seccmp -h >> output.txt
seccmp scm-07 mcmillan serban /dev/sda CC /dev/hdb 7F -h >>
output.txt
```

C4.7.5 Expected results

seccmp should issue its usage mode in each of the four cases.

C5 *partcmp* Test Case Specifications

C5.1 pcm-01

C5.1.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and the selected hard disk drives. Specifically, we want to test whether ***partcmp*** creates a new log file with the default name when no log file is present, correctly logs the partitions, prompts the user for the partition indexes, logs a one-word comment entered on the command line, the program execution, and the comparison result when both partitions are of type primary FAT32, the source partition is smaller than the destination partition and the two partitions have the same contents on the length of the smaller one.

C5.1.2 Test setup

We present here the setup for this test case and other cases as well.
Select two hard disk drives, for example, the IDE hard disk labeled "7F" and the SCSI hard disk labeled "CC". Mount the disks in the test computer (e.g., McMillan) as DOS drives 81 and 82 respectively. When we run Linux, the corresponding devices will be /dev/hdb and /dev/sda respectively. Insert the CD containing the FS-TST v1.0 tools in the CD drive. Boot up the computer from the FS-TST v1.0 boot diskette.

Use FS-TST v1.0 ***diskwipe*** tool to initialize the disk "7F" with 0x7F and the disk "CC" with 0xCC. Use ***diskchg*** to zero the first sector of each disk. Use PartitionMagic to create the following partitions:

On the disk labeled "7F" (mounted as DOS drive 81):
- a primary partition FAT32 of about 9GB;
- a primary partition Linux Ext2 of about 11GB;
- a logical partition FAT32 of about 203MB;
- a logical partition FAT16 of about 203MB.

On the disk labeled "CC" (mounted as DOS drive 82):
- a primary partition FAT32 of about 10GB;
- a primary partition Linux Ext2 of about 10GB;
- a logical partition FAT32 of the same size as the logical FAT32 on disk "7F";
- a logical partition FAT16 of about 305MB.

Use FS-TST v1.0 *seccopy* to copy:
- the primary FAT32 on "7F" to the primary FAT32 on "CC";
- the primary Linux Ext2 on "CC" to the primary Linux Ext2 on "7F";
- the logical FAT32 on "7F" to the logical FAT32 on "CC";
- the logical FAT16 on "7F" to the logical FAT16 on "CC".

(Alternatively, you could reboot to Linux and use the *dd* tool to copy the partitions.)

Note: You can extract the LBA start address and the length of each partition by using the PartitionMagic's "Info" menu or the *partab* tool.

Reboot to Linux. The disk drives “7F” and “CC” should be recognized as Linux devices /dev/hdb and /dev/sda respectively.

C5.1.3 Test case dependencies

None.

C5.1.4 Procedure

Consider disk “7F” (/dev/hdb) as source and disk “CC” (/dev/sda) as destination. Run ***partcmp*** to compare the source primary FAT32 partition to the destination primary FAT32 partition (this is the case source smaller than destination, but with the same contents on the smaller length). Use the *-comment* option with one-word comment, and interactive selection of partitions. Use the same fill values as when we initialized the disks at setup time.

```
partcmp pcm-01 mcmillan serban /dev/hdb 7F /dev/sda CC -
comment CompareLargeFAT32
```

When ***partcmp*** prompts the user for partition indexes, enter the indexes for the two primary FAT32 partitions (extract the indexes from the partition tables displayed by ***partcmp***.)

Use the *ls* command and a text editor to examine the log file’s existence, name, and contents. Verify the count of equal and different sectors reported by ***partcmp***.

Note. ***partcmp*** should report the type of in-excess sectors of the destination partition (whether they’re diskwipe-style filled, or zero-filled, etc.) You could run ***diskwipe*** on each disk as part of the test setup to have each sector diskwipe-style filled, then zero a few of them, or fill a few of them with another value by using the ***diskchg*** tool.

C5.1.5 Expected results

partcmp creates a log file with the default name “cmpptlog.txt” on the log disk (which is the Linux OS disk).
partcmp logs the comment and the correct information required by features 1, 2, 3.
partcmp correctly displays the entries in each partition table.
partcmp logs the partitions to be compared, and the number and range of different and equal sectors of the partitions. For the destination partition, which is larger than the source partition, ***partcmp*** correctly categorizes the in-excess sectors.

C5.2 pcm-02

C5.2.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and the selected hard disk drives. Specifically, we want to test whether ***partcmp*** creates a new log file when using the *-new_log* option although a log file with the same default name exists, logs a multi-word comment entered on the command line, correctly logs the partitions, prompts the user for the partition indexes, logs the program execution and the comparison result, when both partitions are of type primary FAT32, the source partition

is bigger than the destination partition, and they have *almost* the same contents on the length of the smaller one. We also want to compare the partitions' boot tracks, by using the -boot option.

C5.2.2 Test setup

Use the setup of pcm-01.

C5.2.3 Test case dependencies

pcm-01. Do not delete the log file created in the previous test case.

C5.2.4 Procedure

Consider disk "CC" (/dev/sda) as the source disk, and "7F" (/dev/hdb) as the destination disk.
Run ***diskchg*** to zero, or to fill in diskwipe-style, or to change (write), a few sectors in the primary FAT32 partition of the source disk (/dev/sda).

```
diskchg pcm-02 mcmillan serban /dev/sda -fill 1000 1000 0
AA
diskchg pcm-02 mcmillan serban /dev/sda -zero 2000
diskchg pcm-02 mcmillan serban /dev/sda -write 3000 30 AA
```

Run ***partcmp*** on the same partitions as in the case pcm-01, but reversing the roles of the source and destination partitions. Use -boot to also compare the boot sectors, *-comment* with a multi-word comment, and -select (we assume that the two primary FAT32 partitions have both index 1):

```
partcmp pcm-02 mcmillan serban /dev/sda CC /dev/hdb 7F -
select 1 1 -boot -comment "Compare FAT32 slightly
different, src > dst" -new_log
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Verify the count of equal and different sectors reported by ***partcmp***.

C5.2.5 Expected results

partcmp creates a new log file with the name "cmpptlog.txt" on the log disk, even though a file with the same name already exists.
partcmp logs the multi-word comment and the correct information required by features 1, 2, and 3.
partcmp logs the partitions to be compared, and the number and range of different and equal sectors of the partitions, including the boot sectors.

C5.3 pcm-03

C5.3.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and the selected hard disk drives. Specifically, we want to test whether ***partcmp*** appends the log record to the existing log file, correctly logs the partitions, prompts the user for the

partition indexes, prompts the user to enter a comment, logs the program execution and the comparison result, when both partitions are of type primary Linux Ext2, the source is larger than the destination, and they have the same contents on the smaller length.

C5.3.2 Test setup

Use the setup of test case pcm-01.

C5.3.3 Test case dependencies

pcm-02. Do not delete the log file created in that test case.

C5.3.4 Procedure

Run ***partcmp*** to compare the primary Linux Ext2 partitions, the source being on disk "7F":

```
partcmp pcm-03 mcmillan serban /dev/hdb 7F /dev/sda CC -boot
```

When prompted, enter the indexes for the Linux Ext partitions. Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Verify the count and range of equal and different sectors reported by ***partcmp***.

C5.3.5 Expected results

partcmp appends the log records to the log file with the name "cmpptlog.txt" created in the previous case.
partcmp prompts the user for a comment, logs the comment and the correct information required by features 1, 2, 3.
partcmp logs the partitions to be compared, and the number and range of different and equal sectors of the partitions and their boot sectors.

C5.4 pcm-04

C5.4.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and the selected hard disk drives. Specifically, we want to test whether ***partcmp*** creates a log file with an alternate name specified in the *-log_name* option, and correctly compares two logical FAT32 partitions with the same size and contents.

C5.4.2 Test setup

Use the setup of case pcm-01.

C5.4.3 Test case dependencies

None.

C5.4.4 Procedure

Run ***partcmp*** to compare the logical FAT32 partitions, the source being on the disk "7F" (/dev/hdb):

```
partcmp pcm-04 mcmillan serban /dev/hdb 7F /dev/sda CC -
boot -log_name pcmlog.txt
```

When prompted, enter the indexes for the logical FAT32 partitions (in our example, 4 and 4).
Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Verify the count of equal and different sectors reported by ***partcmp***.

C5.4.5 Expected results

partcmp creates a new log file with the name "pcmlog.txt".
partcmp prompts the user for a comment, logs the comment and the correct information required by features 1, 2, 3.
partcmp logs the partitions to be compared, and the number and range of different and equal sectors of the partitions. Actually, all sectors except the boot track sectors should compare equal.

C5.5 pcm-05

C5.5.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and the selected hard disk drives. Specifically, we want to test whether ***partcmp*** appends the log records to a log file with an alternate name specified in the *-log_name* option, and whether it correctly compares two logical FAT32 partitions with the same size and a few different sectors.

C5.5.2 Test setup

Use the setup of pcm-01.

C5.5.3 Test case dependencies

pcm-04, in order to test appending records to the log file with alternate name created in the previous case.

C5.5.4 Procedure

We will again compare the logical FAT32 partitions, keeping disk "7F" (/dev/hdb) as source disk.
Run ***diskchg*** to fill a few sectors of the source logical FAT32 partition with some value (not 0x7F) in diskwipe-style:

```
diskchg pcm-05 mcmillan serban /dev/hdb -fill 40966813
40966813 0 AA
diskchg pcm-05 mcmillan serban /dev/hdb -fill 40966813
40967813 0 AA
diskchg pcm-05 mcmillan serban /dev/hdb -fill 40966813
40968813 0 AA
diskchg pcm-05 mcmillan serban /dev/hdb -fill 40966813
40969813 0 AA
```

Run ***partcmp*** to compare the logical FAT32 partitions, the source being on the disk "7F" (/dev/hdb):

```
partcmp pcm-05 mcmillan serban /dev/hdb 7F /dev/sda CC -
log_name pcmlog.txt -boot
```

When prompted, enter the indexes for the logical FAT32 partitions (in our example, 4 and 4).
Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Verify the count of equal and different sectors reported by ***partcmp***.

C5.5.5 Expected results

partcmp appends the log records to the log file with the alternate name "pcmlog.txt" created in the previous test case.
partcmp prompts the user for a comment, logs the comment and the correct information required by features 1, 2, 3.
partcmp logs the partitions to be compared, and the number and range of different and equal sectors of the partitions. Actually, all sectors should compare equal, except the four diskwipe-style filled and the boot track sectors.

C5.6 pcm-06

C5.6.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and the selected hard disk drives. Specifically, we want to test whether ***partcmp*** creates a new log file with an alternate name specified in the *-log_name* option although such file already exists, and whether it correctly compares two logical FAT16 partitions with the source size less than the destination, and the same contents on the smaller length.

C5.6.2 Test setup

Use the setup of pcm-01.

C5.6.3 Test case dependencies

pcm-05; do not delete the log file created in that test case.

C5.6.4 Procedure

Run ***partcmp*** to compare the logical FAT16 partitions, using the "7F" (/dev/hdb) disk as source:

```
partcmp pcm-06 mcmillan serban /dev/hdb 7F /dev/sda CC -
boot -log_name pcmlog.txt -new_log
```

When prompted, enter the indexes for the logical FAT16 partitions (which in our example are 6 and 6).
Use the *ls* command and a text editor to examine the log file's existence, name, and contents.

C5.6.5 Expected results

partcmp creates a new log file with the alternate name “pcmlog.txt”, even though a file with the same name already exists.
partcmp prompts the user for a comment, logs the comment and the correct information required by features 1, 2, 3.
partcmp logs the partitions to be compared, and the number and range of different and equal sectors of the partitions. Actually, all sectors of the source partition should compare equal to the destination’s sectors. ***partcmp*** should categorize the destination’s in-excess sectors.

C5.7 pcm-07

C5.7.1 Purpose

The purpose of this test case is to test whether ***partcmp*** correctly detects invalid partition indexes, for example indexes that point to empty partition table entries.

C5.7.2 Test setup

Use the setup of pcm-01.

C5.7.3 Test case dependencies

None.

C5.7.4 Procedure

Run ***partcmp*** and use the indexes 8 and 8, which point to empty partition table entries:

```
partcmp pcm-07 mcmillan serban /dev/hdb 7F /dev/sda CC -boot -new_log -select 8 8
```

Use the *ls* command and a text editor to examine the log file’s existence, name, and contents.

C5.7.5 Expected results

partcmp detects the indexes point to empty entries, logs an error message, and terminates execution.

C5.8 pcm-08

C5.8.1 Purpose

The purpose of this test case is to test whether ***partcmp*** correctly detects invalid partition indexes, for example indexes that do not point to any partition.

C5.8.2 Test setup

Use the setup of pcm-01.

C5.8.3 Test case dependencies

None.

C5.8.4 Procedure

Run ***partcmp*** and use the indexes 9 and 9, which do not point to any partition:

```
partcmp pcm-08 mcmillan serban /dev/hdb 7F /dev/sda CC -
boot -new_log -select 9 9
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents.

C5.8.5 Expected results

partcmp detects the indexes are invalid and terminates execution.

C5.9 pcm-09

C5.9.1 Purpose

The purpose of this test case is to test whether ***partcmp*** displays its usage when prompted so by the *-h* option, or when invoked without arguments or with incorrect arguments.

C5.9.2 Test setup

None.

C5.9.3 Test case dependencies

None.

C5.9.4 Procedure

Run ***partcmp*** as shown below and capture its standard output into a file:

```
partcmp > output.txt
partcmp -h >> output.txt
partcmp pcm-09 mcmillan serban -h >> output.txt
partcmp pcm-09 mcmillan serban /dev/hdb 7F /dev/sda CC -h>>
output.txt
```

Use the *ls* command and a text editor to examine the content of the output.txt file.

C5.9.5 Expected results

In each run case, ***partcmp*** displays its usage mode.

C6 *diskcmp* Test Case Specifications

C6.1 dcm-01

C6.1.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and the selected IDE/SCSI hard disk drives. Specifically, we want to test whether *diskcmp* creates a new log file with the default name when no log file is present, correctly logs the drives, a one-word comment entered on the command line, the program execution, and the comparison result when the source disk size is bigger than the destination disk size and the disks have the same contents on the smaller size. Before running *diskcmp*, we will copy the first *n* sectors of the source disk to the destination disk starting from address 0, where *n* is the total number of sectors of the destination disk (the smaller one).

C6.1.2 Test setup

Use "McMillan" as test computer, for example.
Select and insert an IDE hard disk (for example "7C") as source and a SCSI hard disk (for example "CC") as destination, such that the source disk has a bigger size than the destination disk. Assume that the disks will be recognized as Linux devices /dev/hdb and /dev/sda respectively.
Reboot to Linux from the disk containing the Linux OS and the FS-TST v2.0 tools. Same disk will be used as log disk. Use the *dd* command to copy the source disk contents onto the destination disk on the smaller length:

```
dd bs=512 count=71687370 if=/dev/hdb of=/dev/sda
```

We assumed that the destination disk "CC" has a smaller size, 71687370 sectors, than the source disk "7F".

C6.1.3 Test case dependencies

None.

C6.1.4 Procedure

Delete all log files from the log disk. Run *diskcmp* to compare the two disks, using the *-comment* option with a one-word comment:

```
diskcmp dcm-01 mcmillan serban /dev/hdb 7F /dev/sda CC -
comment DiskComparison
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Verify the count of equal and different sectors reported by *diskcmp*.

C6.1.5 Expected results

diskcmp creates a (new) log file with the name "cmplog.txt" on the log disk, logs the comment and the other information required by features 1, 2, 3.

diskcmp logs the hard drives to be compared, and the number and range of different and equal disk sectors. Actually, *diskcmp* should report 71687370 equal sectors, no different sectors, and in-excess sectors on the source disk.

C6.2 dcm-02

C6.2.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and the selected hard disk drives. Specifically, we want to test whether *diskcmp* appends the log records to an existing log file with the default name, correctly logs the drives, a multi-word comment entered on the command line, the program execution, and the comparison result when the source disk is smaller than the destination disk and the disks have *almost* the same contents on the smaller length.

C6.2.2 Test setup

Use the setup of test case dcm-01, with the difference that the SCSI disk "CC" will be the source drive, and the IDE disk "7F" will be the source drive.

C6.2.3 Test case dependencies

dcm-01. Do not delete the log file created in the previous case.

C6.2.4 Procedure

Use the ***diskchg*** tool to modify a few sectors on the source disk "CC" (known as device /dev/sda), among them the first and the last sectors of the disk. For example, fill those sectors with 0xAA in diskwipe-style:

```
diskchg dcm-02 mcmillan serban /dev/sda -fill 0 0 0 AA
diskchg dcm-02 mcmillan serban /dev/sda -fill 1000000
1000000 0 AA
diskchg dcm-02 mcmillan serban /dev/sda -fill 2000000
2000000 0 AA
diskchg dcm-02 mcmillan serban /dev/sda -fill 71687369
71687369 0 AA
```

Run *diskcmp* to compare the two disks, using the *-comment* option with a multi-word comment:

```
diskcmp dcm-02 mcmillan serban /dev/sda CC /dev/hdb 7F -
comment "Compare disks, src<dst, almost equal contents,
append log"
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Verify the count of equal and different sectors reported by *diskcmp*.

C6.2.5 Expected results

diskcmp appends the log records to the existing log file "cmplog.txt", logs the multi-word comment provided on the command line, logs the correct information required by

features 1, 2, 3. *diskcmp* logs the hard drives to be compared, and the number and range of different and equal sectors of the disks. Actually, in our setup and example, it should report only 4 sectors different, and the rest of 71687366 equal, and it should categorize the in-excess sectors of the destination drive: source byte filled, destination byte filled, other byte filled, zero filled, and other.

C6.3 dcm-03

C6.3.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and two hard disk drives with the same size, filled with the same value in diskwipe-style, and only a handful of different sectors at known addresses. Among other things, we will test whether *diskcmp* will create a new log file when instructed by the option *-new_log*, although a file with the same name already exists, and whether it allows the user to enter a comment interactively.

C6.3.2 Test setup

Use “McMillan” as test computer, for example.
Select two (IDE) hard disks with the same capacity (e.g., the hard disk drives labeled “82” and “80”) and mount them so that they will be recognized as devices /dev/hdb and /dev/hdd. Reboot to Linux from the hard disk containing the Linux OS and the FS-TST v2.0 tools. The same disk will also be used as log disk.

C6.3.3 Test case dependencies

dcm-02. Do not delete the log file created in the previous case.

C6.3.4 Procedure

Assuming both the source and destination drives use the same geometry (the number of tracks/cylinder), use ***diskwipe*** to initialize both the source and destination disks with the same value 0x82:

```
diskwipe dcm-03 mcmillan serban /dev/hdb 82 -src
diskwipe dcm-03 mcmillan serban /dev/hdd 82 -dst
```

If the geometry is different, use the *-heads* option with the same value for both disks.

Run ***diskchg*** a few times to fill, write, or zero a few sectors, among them the first and the last:

```
diskchg dcm-03 mcmillan serban /dev/hdd -fill 0 0 0 AA
diskchg dcm-03 mcmillan serban /dev/hdd -write 156301487
511 AA
diskchg dcm-03 mcmillan serban /dev/hdd -zero 100000000
```

Run *diskcmp* to compare the two disks, using the *-new_log* option:

```
diskcmp dcm-03 mcmillan serban /dev/hdb 82 /dev/hdd 82 -
new_log
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Verify the count of equal and different sectors.

C6.3.5 Expected results

diskcmp prompts the user for a comment, creates a new log file "cmplog.txt", and logs the comment. It logs the correct information required by features 1, 2, 3.
diskcmp logs the hard drives to be compared, and the number of equal and different sectors. Actually, in our example of setup, only 3 sectors should be found different.

C6.4 dcm-04

C6.4.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and selected hard disk drives with the same size, when the disks are filled in ***diskwipe***-style with different values, and only a handful of sectors are equal. Among other things we will test whether *diskcmp* creates a log file with an alternate name when we use the *-log_name* option.

C6.4.2 Test setup

Mount two IDE disks with the same size ("82" and "80" respectively) in slots of the "McMillan" computer, so that they will be recognized as devices /dev/hdb, /dev/hdd respectively. Reboot to Linux.

C6.4.3 Test case dependencies

None.

C6.4.4 Procedure

Wipe out the disks with different values, using the same geometry with 255 heads/cylinder:

```
diskwipe dcm-04 mcmillan serban /dev/hdb 82 -heads 255 -
noask -new_log -src
diskwipe dcm-04 mcmillan serban /dev/hdd 80 -heads 255 -
noask -new_log -dst
```

Fill some sectors of the destination disk with the same value as the source disk, using the -fill function of the ***diskchg*** tool and the same geometry as before:

```
diskchg dcm-04 mcmillan serban /dev/hdd -fill 1000000
1000000 255 82
diskchg dcm-04 mcmillan serban /dev/hdd -fill 2000000
2000000 255 82
```

```
diskchg dcm-04 mcmillan serban /dev/hdd -fill 3000000
3000000 255 82
```

Run *diskcmp* with the *-log_name* option to compare the disks:

```
diskcmp dcm-04 mcmillan serban /dev/hdb 82 /dev/hdd 80 -
log_name diskcmplog.txt
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Verify the count of equal and different sectors. Verify whether *diskcmp* displays any information about sectors filled in ***diskwipe***-style.

C6.4.5 Expected results

diskcmp prompts the user for a comment, creates a new log file "diskcmplog.txt", and logs the comment. It logs the correct information required by features 1, 2, 3.
diskcmp logs the hard drives to be compared, and the number of equal and different sectors. Actually, only 3 sectors should match.

C6.5 dcm-05

C6.5.1 Purpose

The purpose of this test case is to test feature 3 on the target computer, namely whether *diskcmp* displays its usage mode when prompted by the *-h* option on the command line.

C6.5.2 Test setup

None.

C6.5.3 Test case dependencies

None.

C6.5.4 Procedure

Run *diskcmp* without arguments, with the *-h* option alone on the command line, with incorrect arguments, and with correct arguments and the *-h* option on the command line, and capture the standard output into a file:

```
diskcmp > output.txt
diskcmp -h >> output.txt
diskcmp dcm-05 mcmillan serban /dev/sda -logname >>
output.txt
diskcmp dcm-05 mcmillan serban /dev/sda CC /dev/hdb 7F -
new_log -h >> output.txt
```

Use the *ls* command and a text editor to examine the log file contents.

C6.5.5 Expected results

diskcmp displays its usage mode in each case.

C7 *corrupt* Test Case Specifications

C7.1 cor-01

C7.1.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and selected hard disk drive. Specifically, we want to test whether ***corrupt*** creates a new log file with the default name when no log file is present, correctly logs a one-word comment entered on the command line, alters the first byte of the image file, logs the program execution and the original and new byte values.

C7.1.2 Test setup

On the test computer, mount an IDE hard disk drive, large enough to accommodate an image file in one of its IDE slots. This disk will be used as a media disk. In the test cases that are described below, we assume that the computer is "McMillan", the media disk is the one labeled "80", and it is mounted so that the corresponding Linux device is /dev/hdd and the DOS drive is 82.
Boot to DOS and use PartitionMagic to create a Linux Ext2 partition on the media disk.
Reboot to Linux. According to our assumptions, the Ext2 partition will be seen as /dev/hdd1. Mount this partition as /media, for example.
Create an image file "imgfile" on /media of a disk, for example of /dev/sda, by using the *dd* command or a custom program, for example:

```
dd bs=512 if=/dev/sda of=/media/imgfile
```

Make a reference copy of the image file called "copy-of-imgfile" in the /media directory by using the *cp* command.

Note. It is possible that the image file is truncated because Linux kernel limits on a file size. In our examples we used an image file of about 17GB.

C7.1.3 Test case dependencies

None.

C7.1.4 Procedure

Run ***corrupt*** to alter the first byte of the image file:

```
corrupt cor-01 mcmillan serban /media/imgfile 0 41 -comment
AlterFirstByte
```

Run the ***cmp*** command to compare the altered image file to the reference copy:

```
cmp -l /media/imgfile /media/copy-of-imgfile > diff.txt
```

Note: the ***cmp*** command outputs the bytes that differ in octal; the byte offset starts from 1.

C7.1.5 Expected results

corrupt creates a new log file with the name “corlog.txt” on the log disk.
corrupt logs the comment and the correct information required by features 1, 3, 4.
The altered image file and the reference copy differ only by the first byte.

C7.2 cor-02

C7.2.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and selected hard disk drive. Specifically, we want to test whether ***corrupt*** appends the log records to the existing log file, correctly logs a multi-word comment entered on the command line, alters the last byte of the image file, logs the program execution and the original and new byte values.

C7.2.2 Test setup

Use the setup of cor-01.

C7.2.3 Test case dependencies

cor-01. Do not delete the previous log file in order to test whether ***corrupt*** appends the log records to the existing log file.

C7.2.4 Procedure

Run ***corrupt*** to alter the last byte of the image file:

```
corrupt cor-02 mcmillan serban /media/imgfile 17247252479
41 -comment "Alter last byte, append log"
```

Run the ***cmp*** command to compare the altered image file to the reference copy:

```
cmp -l /media/imgfile /media/copy-of-imgfile > diff.txt
```

C7.2.5 Expected results

corrupt appends the log records to the previous log file corlog.txt.
corrupt logs the comment and the correct information required by features 1, 3, 4.
The altered image file and the reference copy differ only by the first (because of the test case cor-01) and last byte.

C7.3 cor-03

C7.3.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and the selected hard disk drive. Specifically, we want to test whether ***corrupt*** creates a new log file when instructed by the *-new_log* option, prompts the user to enter a comment, alters a byte somewhere in the middle of an image file, logs the program execution and the original and new byte values.

C7.3.2 Test setup

Use the setup of cor-01.

C7.3.3 Test case dependencies

cor-02. Do not delete the previous log file in order to test whether ***corrupt*** creates a new log file.

C7.3.4 Procedure

Run ***corrupt*** to alter a byte of the image file:

```
corrupt cor-03 mcmillan serban /media/imgfile 10000000000
41 -new_log
```

Run the ***cmp*** command to compare the altered image file to the reference copy:

```
cmp -l /media/imgfile /media/copy-of-imgfile > diff.txt
```

C7.3.5 Expected results

corrupt create a new log file corlog.txt, even though one with the same name existed.
corrupt prompts the user to enter a comment.
corrupt logs the comment, the original and new values of the byte at offset 10000000000.
The altered image file and the reference copy differ only by the bytes modified in the cases cor-01, cor-02, cor-03.

C7.4 cor-04

C7.4.1 Purpose

The purpose of this test case is to test features 1, 2, 3, and 4 on the target computer and the selected hard disk drive. Among other things, we want to test whether ***corrupt*** creates a log file with an alternate name specified in the *-log_name* option.

C7.4.2 Test setup

Use the setup of cor-01.

C7.4.3 Test case dependencies

None.

C7.4.4 Procedure

Run ***corrupt*** to alter a byte of the image file:

```
corrupt cor-04 mcmillan serban /media/imgfile 10000000001
41 -log_name corruptlog
```

C7.4.5 Expected results

corrupt create a new log file corruptlog.txt.
corrupt prompts the user to enter a comment.

corrupt logs the comment, the original and new values of the byte at the specified offset.

C7.5 cor-05

C7.5.1 Purpose

The purpose of this test case is to test whether ***corrupt*** detects that a byte offset is invalid, i.e., outside the image file range.

C7.5.2 Test setup

Use the setup of cor-01.

C7.5.3 Test case dependencies

None.

C7.5.4 Procedure

Run ***corrupt*** and specify a byte offset outside the range of the image file. Capture the stdout:

```
corrupt cor-05 mcmillan serban /media/imgfile 17247252480
41 -new_log
```

Examine the stdout for signs that ***corrupt*** has detected the invalid offset.

C7.5.5 Expected results

corrupt displays a warning about the invalid offset and rejects the request to alter the byte.

C7.6 cor-06

C7.6.1 Purpose

The purpose of this test case is to test whether ***corrupt*** displays its usage mode when instructed by the *-h* option.

C7.6.2 Test setup

None.

C7.6.3 Test case dependencies

None.

C7.6.4 Procedure

Run ***corrupt***: without any argument, with incorrect arguments, with the *-h* option as the only argument on the command line, and with the *-h* option together with other arguments. Capture the standard output into a file:

```
corrupt > output.txt
corrupt cor-06 mcmillan serban /media/imgfile -logname -h
>> output.txt
```

```
corrupt -h >> output.txt
corrupt cor-06 mcmillan serban /media/imgfile 41 -new_log -
h >> output.txt
```

C7.6.5 Expected results

corrupt displays its usage mode on the stdout in each case.

C8 *logsetup* Test Case Specifications

C8.1 lgs-01

C8.1.1 Purpose

The purpose of this test case is to test whether ***logsetup*** correctly logs the information about the setup of a source disk provided by the user in arguments on the command line, namely the hard disk drive, the host computer, the operator, the operating system loaded on the disk, and some options. Each argument is to be interpreted as a character string by ***logsetup***, so either the argument does not contain white space, or it may contain white space and then it must be included in double quotes. ***logsetup*** simply copies the argument to the log file it generates. The only logged information that is not provided by the user is the current time and date.

C8.1.2 Test setup

None.

C8.1.3 Test case dependencies

None.

C8.1.4 Procedure

Run ***logsetup*** with some arguments:

logsetup CC:/dev/sda McMillan serban None

Observe whether the log file contains the arguments provided on the command line.

C8.1.5 Expected results

logsetup creates a new log file with the name “setup.txt” on the log disk. It logs the following information: the hard disk drive (CC:/dev/sda), the host computer (McMillan), the operator (serban), the operating system loaded on the disk (None). It adds the current time and date.

C9 *logcase* Test Case Specifications

C9.1 Lgc-01

C9.1.1 Purpose

The purpose of this test case is to test whether ***logcase*** correctly logs the information about a test case provided by the user in arguments on the command line, namely the case identifier, the host computer, the operator, the three disks that might be used (source, destination, and media - each argument is mandatory). ***logcase*** simply copies the argument to the log file it generates. The only logged information that is not provided by the user is the current time and date.

C9.1.2 Test setup

None.

C9.1.3 Test case dependencies

None.

C9.1.4 Procedure

Run ***logcase***:

logcase pcm-01 McMillan serban CC:/dev/sda 7F:/dev/hdb none

C9.1.5 Expected results

logcase creates a new log file with the name “case.txt” on the log disk. It logs the following information: the test case identifier (pcm-01), the host computer (McMillan), the operator (serban), the source disk (CC:/dev/sda), the destination disk (7F:/dev/hdb), and the media disk (none) used in the test case. It adds the current time and date.

C10 *adjcmp* Test Case Specifications

C10.1 acm-01

C10.1.1 Purpose

The purpose of this test case is to test features 1-6 on the target computer and the selected hard disk drives by specifying the -layout argument. Specifically, we want to test whether ***adjcmp***:

a. Creates a new log file when no log file is present.
b. Logs a one-word comment entered on the command line.
c. Logs the source and destination drives.
d. Logs the program execution.
e. Logs the partitions tables of each disk.
f. Correctly detects the disk layouts and records the location of each disk chunk.

C10.1.2 Test setup

On the test computer (for example "McMillan"), mount the IDE hard disk "7F" in a slot as DOS drive 0x81 (source) and the SCSI hard disk "CC" in a slot as DOS drive 0x82 (destination). Insert the CD containing the FS-TST1.0 tools in the CD drive. Reboot the computer from the FS-TST1.0 boot diskette.
Use PartitionMagic to create the following partitions on the source disk: primary FAT32 (approx. 3000MB), primary Linux Ext2 (approx. 2000MB), logical FAT16 (approx. 200MB), logical FAT32 (approx. 200MB), all separated by unallocated space.
Use PartitionMagic to create the following partitions on the destination disk: primary FAT32 (same size as the primary FAT32 on the source disk), primary Linux Ext2 (same size as the primary Linux Ext2 on the source disk), logical FAT16 (same size as the logical FAT16 on the source disk), logical FAT32 (same size as the logical FAT32 on the source disk), and logical NTFS (approx. 200MB), all separated by unallocated space.

Run ***seccopy*** to copy the source primary FAT32, primary Linux Ext2, logical FAT16, logical FAT32 partitions onto the destination primary FAT32, primary Linux Ext2, logical FAT16, and logical FAT32 partitions respectively.

Reboot to Linux.

C10.1.3 Test case dependencies

None.

C10.1.4 Procedure

Run ***adjcmp***:

```
adjcmp acm-01 mcmillan serban /dev/hdb 7F /dev/sda CC -
layout -comment "Layout"
```

Use the *ls* command and a text editor to examine the presence and content of the log file cmpalog.txt generated by ***adjcmp***.

C10.1.5 Expected results

adjcmp creates a new log file cmpalog.txt. ***adjcmp*** logs the comment, the source and destination drives, the program execution, logs the partition tables of each disk, computes and logs the location, size, and type (boot, partition, unallocated) of each disk chunk.

C10.2 acm-02

C10.2.1 Purpose

The purpose of this test case is to test features 1-6 and 8-11 on the target computer and the selected hard disk drives. We use the *-new_log* option to test whether ***adjcmp*** creates a new log file although a log file with the same name already exists. We use a multi-word comment on the command line. We run ***adjcmp*** on disks where the partitions are separated by unallocated space on both the source and destination disks, so that the first few source and destination chunks of type P (partition) correspond in assignment. Also, we prepared source and destination partitions that compare equal. We also want to test how ***adjcmp*** categorizes in-excess destination chunks.

C10.2.2 Test setup

Use the setup of acm-01.

C10.2.3 Test case dependencies

acm-01, to test whether a new log file is created when the *-new_log* option is used and an old log file exists.

C10.2.4 Procedure

Run ***adjcmp*** with the *-new_log* option and a multi-word comment:

```
adjcmp acm-02 mcmillan serban /dev/hdb 7F /dev/sda CC -
new_log -comment "Compare automatically assigned
partitions"
```

Use the *ls* command and a text editor to examine the presence and content of the log file cmpalog.txt generated by ***adjcmp***.

C10.2.5 Expected results

adjcmp creates a new log file cmpalog.txt. ***adjcmp*** logs the comment, the source and destination drives, the program execution, the partition table entries of each disk, computes the type, location, and size of the disk chunks, assigns the source to destination chunks as expected, compares the chunks and logs the comparison results; given the setup, first 4 partitions should compare equal. As the destination has chunks in excess, ***adjcmp*** categorizes their sectors as described in the features. Finally, it logs a summary of the comparison.

C10.3 acm-03

C10.3.1 Purpose

The purpose of this test case is to test features 1-11 on the target computer and the selected hard disk drives. In particular, we want to test:
- Whether ***adjcmp*** appends the log records to an existing log file.
- Whether ***adjcmp*** prompts the user for a comment when the *-comment* option is not used.
- Whether ***adjcmp*** lets the user to manually assign the source chunks to destination chunks when we use the -assign option.

C10.3.2 Test setup

Use the setup of acm-01.

C10.3.3 Test case dependencies

acm-02. Thus, we will be able to test whether ***adjcmp*** appends the log to the existing log file.

C10.3.4 Procedure

Run ***adjcmp***:

```
adjcmp acm-03 mcmillan serban /dev/hdb 7F /dev/sda CC -assign
```

When prompted, assign the source disk chunks to destination disk chunks:

0 B → 0 B
1 P → 1 P
2 U → 2 U
3 P → 3 P
4 U → 4 U
5 b → 8 b
6 P → 9 P
7 U → 10 U
8 b → 5 b
9 P → 6 P
10 U → 7 U

Use the *ls* command and a text editor to examine the presence and content of the log file cmpalog.txt generated by ***adjcmp***.

C10.3.5 Expected results

adjcmp appends the log records to the existing log file cmpalog.txt created in the previous test case. ***adjcmp*** prompts the user for a comment, logs the comment, the source and destination drives, the program execution, logs the boot tracks, partitions and unallocated space of each disk, prompts the user for chunk assignment, compares the assigned chunks, and logs the results. This time, source chunks 6 and 9 should not compare equal to destination chunks 9 and respectively 6.

C10.4 acm-04

C10.4.1 Purpose

The purpose of this test case is to test features 1-11 on the target computer and the selected hard disk drives. In particular, we want to test whether ***adjcmp*** allows the user to specify an alternate log file name by using the *-log_name* option. We'll use the same disks as before, but reverse the source and destination to test how ***adjcmp*** treats in-excess source chunks. Also, prior to running ***adjcmp*** we modify a few sectors in the partitions that otherwise would have compared equal (we can use ***diskchg*** for that).

C10.4.2 Test setup

Use the setup of case acm-01.
Use ***diskchg*** to change a few sectors in each of the four partitions of the disk "7F".

C10.4.3 Test case dependencies

None.

C10.4.4 Procedure

Run ***adjcmp*** using automatic assignment of disk chunks and the *-log_name* option:

```
adjcmp acm-04 mcmillan serban /dev/sda CC /dev/hdb 7F -
log_name adjcmplog.txt
```

Use the *ls* command and a text editor to examine the presence and content of the log file adjcmplog.txt generated by ***adjcmp***.

C10.4.5 Expected results

adjcmp creates a new log file adjcmplog.txt, prompts the user to enter a comment, logs the comment, the source and destination drives, the program execution, logs the boot tracks, partitions and unallocated space of each disk, automatically assigns the source chunks to destination chunks, compares the assigned disk chunks, logs the comparison results and a summary, as detailed in the features.

C10.5 acm-05

C10.5.1 Purpose

The purpose of this test case is to test features 1-11 on the target computer and the selected hard disk drives, which contain large (>8MB) primary and/or logical partitions. Also, we intend to examine how can the user manually assign source chunks of type P (partitions) to destination chunks P when the source has unallocated space interspersed with the partitions, while the destination does not.

C10.5.2 Test setup

Use a computer with extended BIOS, "McMillan" for example. Mount the IDE hard disk "7F" as DOS drive 0x81 (source). Mount the SCSI hard disk "CC" as DOS drive 0x82 (destination). Insert the CD containing the Forensic Software Testing Support Tools v1.0

in the CD drive. Reboot the computer from the FS-TSTv1.0 boot diskette. Use PartitionMagic to create the following partitions on the source disk "7F": primary FAT32 (10GB), Linux Ext2 (9GB), logical FAT16 (1000MB), logical FAT32 (9GB), all separated through unallocated space.
Use PartitionMagic to create the following partitions on the destination disk "CC": primary FAT32 (9GB), Linux Ext2 (10GB), logical FAT16 (2000MB), and logical FAT32 (8GB), without any unallocated space in-between.

Run ***seccopy*** to copy the smallest partition onto the corresponding partition (e.g., copy the destination primary FAT32 onto the source primary FAT32 partition, etc.)

Reboot to Linux.

C10.5.3 Test case dependencies

None.

C10.5.4 Procedure

Run ***adjcmp***:

```
adjcmp acm-05 mcmillan serban /dev/hdb 7F /dev/sda CC -assign -new_log
```

When prompted, manually assign the chunks as follows, in order to avoid assigning unallocated space to partitions:

0 B → 0 B
1 P → 1 P
2 U → 0 B
3 P → 2 P
4 U → 0 B
5 b → 3 b
6 P → 4 P
7 U → 0 B
8 b → 5 b
9 P → 6 P
10 U → 7 U

Use the *ls* command and a text editor to examine the presence and content of the log file cmpalog.txt generated by ***adjcmp***.

C10.5.5 Expected results

adjcmp creates a new log file cmpalog.txt, prompts the user to enter a comment, logs the comment, the source and destination drives, the program execution, logs the boot tracks, partitions and unallocated space of each disk, prompts the user for chunk assignment, compares the disk chunks according to the assignment, logs the comparison results and a summary, as detailed in the features. When a source chunks is smaller than the

corresponding destination, the excess destination sectors are categorized as detailed in the FS-TST 2.0 documentation.

C10.6 acm-06

C10.6.1 Purpose

The purpose of this test case is to test whether ***adjcmp*** displays its usage mode when invoked with the *-h* option.

C10.6.2 Test setup

None.

C10.6.3 Test case dependencies

None.

C10.6.4 Procedure

Run ***adjcmp*** with the *-h* option and capture its standard output into a file:

```
adjcmp -h > outputlog.txt
adjcmp acm-06 mcmillan serban /dev/hdb 7F /dev/sda CC -h >>
outputlog.txt
```

Use a text editor to examine the content of the outputlog.log file.

C10.6.5 Expected results

adjcmp should display its usage mode only.

C11 *sechash* Test Case Specifications

C11.1 shs-01

C11.1.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and the selected hard disk drive. Specifically, we want to test whether ***sechash*** creates a new log file with the default name for the -before option when no log file is present, correctly logs the hard disk drive, a one-word comment entered on the command line, and the program execution. Also, we want to test whether omitting -first and -last options defaults to hashing the entire disk, and whether by default ***sechash*** computes the SHA-1 hash of the requested block of sectors.

C11.1.2 Test setup

Select and insert the target disk drive into a slot of the test computer. For example, assume that the target disk is the one externally labeled "CC" and it will be the Linux device /dev/sda. Boot up to Linux. Use the script cal-drive.csh available from NIST's "Computer Forensic Reference Data Set" (CFReDS) project to write a certain pattern on device /dev/sda whose SHA-1 and MD5 hashes are known. Invoke the cal-drive.csh script as follows:

```
cal-drive.csh sda
```

Save the log file cal-log.txt written by the script for later comparisons with the hashes computed by ***sechash***.

C11.1.3 Test case dependencies

None.

C11.1.4 Procedure

Delete all log files from the log disk (the Linux boot disk).

Run ***sechash*** using a one-word comment and the -before option. Omit the -first, -last, and -*hash* options:

```
sechash.csh shs-01 mcmillan serban /dev/sda CC -before -
comment HashEntireDisk
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Compare the SHA-1 hash value computed by ***sechash*** to the hash value computed by the cal-drive.csh script.

C11.1.5 Expected results

sechash creates a log file with the name "hashbsec.txt" on the log disk. ***sechash*** logs the comment, computes the SHA-1 hash of the entire disk, and logs the correct information required by features 1, 2, 4, 5.

C11.2 shs-02

C11.2.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and selected hard disk drive. Specifically, we want to test whether ***sechash*** appends the log records to an existing log file, correctly logs a multi-word comment entered on the command line, and the program execution. Also, we want to test whether ***sechash*** correctly computes the MD5 hash of the entire disk specified as a block of sectors.

C11.2.2 Test setup

Use the setup of test case shs-01. Do not delete the log file created by test case shs-01.

C11.2.3 Test case dependencies

shs-01.

C11.2.4 Procedure

Run this test case right after shs-01.

Run ***sechash*** using the *-comment* option with a multi-word comment, the -before option, the -first and -last options specifying the first and last sector of the disk (you may extract this information from the drive information logged in the previous case), and the *-hash* option with the md5 function:

```
sechash.csh shs-02 mcmillan serban /dev/sda CC -before -
first 0 -last 71687369 -comment “Hash entire disk” -hash
md5sum
```

Use the *ls* command and a text editor to examine the log file’s existence, name, and contents. Compare the MD5 hash value computed by ***sechash*** to the hash value computed by the cal-drive.csh script.

C11.2.5 Expected results

sechash appends the log records to the log file “hashbsec.txt” created in the preceding test case. ***sechash*** logs the comment, computes the correct MD5 hash for the entire disk, and logs the correct information required by features 1, 2, 4, and 5.

C11.3 shs-03

C11.3.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, 5 on the target computer and selected hard disk drive. Specifically, we want to test whether ***sechash*** creates a log file with the name characteristic for the -after option. Also, we want to verify that ***sechash*** correctly computes the SHA-1 value of a modified disk contents. In order to do that, we could use exactly the same modification we did for test case dhs-04 of ***diskhash***, assuming we had run that test case before the current one, and in both cases the script cal-drive.csh has written the same pattern on the selected drive (actually we could run the test cases dhs-04 and shs-03 consecutively, without modifying the disk in between).

C11.3.2 Test setup

Use the setup of test case dhs-04.

C11.3.3 Test case dependencies

dhs-04 (for the setup and change to the disk contents, and the SHA-1 hash value).

C11.3.4 Procedure

Use ***diskchg*** to modify the last byte of the last sector of the selected disk in exactly the same way as we did in test case dhs-04:

```
diskchg shs-03 mcmillan serban /dev/sda -write 71687369 511
46
```

Run ***sechash*** without the *-comment* option, but with *-new_log*, *-after*, *-first*, *-last*, and *-hash* options, specifying the entire disk and SHA-1 hash:

```
sechash.csh shs-03 mcmillan serban /dev/sda CC -new_log
-after -first 0 -last 71687369 -hash sha1sum
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Compare the SHA-1 hash value computed by ***sechash*** with the hash value computed by ***diskhash*** in the dhs-04 test case.

C11.3.5 Expected results

sechash creates a new log file "hashasec.txt", prompts the user for a comment, computes the correct SHA-1 hash, and logs the required information in the log file.

C11.4 shs-04

C11.4.1 Purpose

The purpose of this test case is exactly as that of the previous case, except that we compute the MD5 hash value of the entire disk after the modification described in shs-03. We also verify that ***sechash*** creates a new log file although one with the same name already exists, when we use the *-new_log* option.

C11.4.2 Test setup

Use the setup and disk content modification of case shs-03 (i.e., dhs-04).

C11.4.3 Test case dependencies

dhs-04 (for the setup and change to the disk contents), dhs-05 (for the MD5 hash value), shs-03 (for the log file).

C11.4.4 Procedure

Run ***sechash*** using a command similar to the one used in the previous case, but specifying the md5sum hash function:

```
sechash.csh shs-04 mcmillan serban /dev/sda CC -new_log -
after -first 0 -last 71687369 -hash md5sum
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Compare the hash value computed in this case to the hash value computed in the test case dhs-05.

C11.4.5 Expected results

sechash creates a new log file "hashasec.txt" although one exists, prompts the user for a comment, computes the correct MD5 hash of the entire disk, and logs the required information in the log file.

C11.5 shs-05

C11.5.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and selected hard disk drive. Specifically, we want to test whether ***sechash*** correctly computes the SHA-1 hash of sector 0, and whether it creates a log file with an alternate name specified by using the *-log_name* option.

C11.5.2 Test setup

Select and insert the target disk drive into a slot of the test computer. For example, assume that the target disk is the one externally labeled "CC" and it will be recognized as the Linux device /dev/sda. Boot up to Linux. Use the script cal-drive-count.csh tailored from the script cal-drive.csh of NIST's "Computer Forensic Reference Data Set" (CFReDS) project, to write a certain pattern on sector 0 of the device /dev/sda whose SHA-1 and MD5 hashes are known. Invoke the cal-drive-count.csh script and save the output for later comparisons as follows:

```
cal-drive-count.csh sda 1 > output.txt
```

Save the log file cal-log.txt written by the script for later comparisons with the hashes computed by ***sechash***.

C11.5.3 Test case dependencies

None.

C11.5.4 Procedure

Run ***sechash*** using the *-log_name* to enter an alternate log file name, -first, and -last options to define the group consisting only of sector 0, and the -hash option to specify the SHA-1 hash to be computed:

```
sechash.csh shs-05 mcmillan serban /dev/sda CC -log_name
sechashlog.txt -first 0 -last 0 -hash sha1sum
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Compare the SHA-1 hash value computed by ***sechash*** to the value computed by the cal-drive-count.csh script.

C11.5.5 Expected results

sechash creates a new log file of name "sechashlog.txt".
sechash prompts the user for a comment, which is correctly logged.
sechash computes the correct SHA-1 hash of sector 0.
sechash logs the correct information required by features 2, 4, 5.

C11.6 shs-06

C11.6.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and selected hard disk drive. Specifically, we want to test whether ***sechash*** correctly computes the MD5 hash of sector 0, and whether it creates a new log file with an alternate name specified by using the *-log_name* option, although one with the same name exists.

C11.6.2 Test setup

Use the same setup as in shs-05.

C11.6.3 Test case dependencies

shs-05 (for the alternate log file to exist).

C11.6.4 Procedure

Run ***sechash*** using the following command:

```
sechash.csh shs-06 mcmillan serban /dev/sda CC -log_name
sechashlog.txt -new_log -first 0 -last 0 -hash md5sum
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Compare the MD5 hash value computed by ***sechash*** to the value computed by the cal-drive.csh script.

C11.6.5 Expected results

sechash creates a new log file with the name "sechashlog.txt", although one with the same name still exists.
sechash prompts the user for a comment, which is correctly logged.
sechash computes the correct MD5 value of the first disk sector, and logs the information required by features 2, 4, 5.

C11.7 shs-07

C11.7.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and selected hard disk drive. Specifically, we want to test whether ***sechash*** correctly computes the SHA-1 hash of the last sector of the hard disk drive.

C11.7.2 Test setup

Select and insert the target disk drive into a slot of the test computer. For example, assume that the target disk is the one externally labeled "CC" and it will be recognized as the Linux device /dev/sda. Use the script cal-drive-count-seek.csh tailored from the script cal-drive.csh of NIST's "Computer Forensic Reference Data Set" (CFReDS) project, to write a certain pattern with known SHA-1 and MD5 hashes on the last sector of the device /dev/sda. Invoke the cal-drive-count-seek.csh script as follows:

```
cal-drive-count-seek.csh sda 1 71687369
```

(71687369 is the LBA address of the last sector on disk "CC"). Save the log file cal-log.txt written by the script for later comparisons with the hashes computed by ***sechash***.

C11.7.3 Test case dependencies

None.

C11.7.4 Procedure

Run ***sechash*** using the -before, -first, -last, *-new_log* options, to compute the SHA-1 hash of the last sector:

```
sechash.csh shs-07 mcmillan serban /dev/sda CC -before -
new_log -first 71687369 -last 71687369
```

Use the *ls* command and a text editor to examine the name and contents of the log file created by ***sechash***.

C11.7.5 Expected results

sechash creates a new log file "hashbsec.txt".
sechash prompts the user for a comment, which is correctly logged.
sechash computes the correct SHA-1 hash of the last sector (equal to the hash value computed by the script cal-drive-count-seek.csh).
sechash logs the correct information required by features 2, 4, 5.

C11.8 shs-08

C11.8.1 Purpose

The purpose of this test case is similar to that of previous test case, the only difference being that we compute the MD5 hash of the last sector.

C11.8.2 Test setup

Use the setup of shs-07.

C11.8.3 Test case dependencies

None.

C11.8.4 Procedure

Run ***sechash*** as in the previous case, but specify the MD5 function instead of SHA-1:

```
sechash.csh shs-08 mcmillan serban /dev/sda CC -before -
new_log -first 71687369 -last 71687369 -hash md5sum
```

C11.8.5 Expected results

sechash creates a new log file "hashbsec.txt".
sechash prompts the user for a comment, which is correctly logged.
sechash computes the correct MD5 hash of the last sector (equal to the hash value computed by the script cal-drive-count-seek.csh).
sechash logs the correct information required by features 2, 4, 5.

C11.9 shs-09

C11.9.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and selected hard disk drive. Specifically, we want to test whether ***sechash*** correctly computes the SHA-1 hash of a group of contiguous sectors.

C11.9.2 Test setup

Select and insert the target disk drive into a slot of the test computer. For example, assume that the target disk is the one externally labeled "CC" and it will be recognized as the Linux device /dev/sda. Use the script cal-drive-count-seek.csh tailored from the script cal-drive.csh of NIST's "Computer Forensic Reference Data Set" (CFReDS) project, to write a certain pattern with known SHA-1 and MD5 hashes on a group of sectors defined by the LBA address of its first sector and the number of sectors. Invoke the cal-drive-count-seek.csh script and save the output for later comparisons as follows:

```
cal-drive-count-seek.csh sda 1000000 10000 > output.txt
```

(in our example the group has 1,000,000 sectors starting at address 10,000). Save the log file cal-log.txt written by the script for later comparisons with the hashes computed by ***sechash***.

C11.9.3 Test case dependencies

None.

C11.9.4 Procedure

Run ***sechash*** using the -before, -first, -last, *-new_log* options, to compute the SHA-1 hash of the sector group:

```
sechash.csh shs-09 mcmillan serban /dev/sda CC -before -
new_log -first 10000 -last 1009999
```

Use the *ls* command and a text editor to examine the name and contents of the log file created by ***sechash***.

C11.9.5 Expected results

sechash creates a new log file “hashbsec.txt”.
sechash prompts the user for a comment, which is correctly logged.
sechash computes the correct SHA-1 hash of the specified group (equal to the hash value computed by the script cal-drive-count-seek.csh).
sechash logs the correct information required by features 2, 4, 5.

C11.10 shs-10

C11.10.1 Purpose

The purpose of this test case is the same as that of the previous case, except that we want to compute the MD5 hash of a group of contiguous sectors.

C11.10.2 Test setup

shs-09.

C11.10.3 Test case dependencies

None.

C11.10.4 Procedure

Run ***sechash*** using a similar command as in shs-09, but specifying the MD5 hash function instead of SHA-1:

```
sechash.csh shs-10 mcmillan serban /dev/sda CC -before -
new_log -first 10000 -last 1009999 -hash md5sum
```

Use the *ls* command and a text editor to examine the name and contents of the log file created by ***sechash***.

C11.10.5 Expected results

sechash creates a new log file “hashbsec.txt”.
sechash prompts the user for a comment, which is correctly logged.
sechash computes the correct MD5 hash of the specified group (equal to the hash value computed by the script cal-drive-count-seek.csh).
sechash logs the correct information required by features 2, 4, 5.

C11.11 shs-11

C11.11.1 Purpose

The purpose of this test case is to verify whether ***sechash*** detects that the -first value is larger than the -last value.

C11.11.2 Test setup

Any disk could be used.

C11.11.3 Test case dependencies

None.

C11.11.4 Procedure

Run ***sechash*** using a -first value larger than the -last value:

```
sechash.csh shs-11 mcmillan serban /dev/sda CC -before -new_log -first 10000 -last 9999 -hash md5sum
```

Examine the standard output and/or the log file, if created, looking for an error message.

C11.11.5 Expected results

sechash issues an error message regarding incorrect LBA addresses.

C11.12 shs-12

C11.12.1 Purpose

The purpose of this test case is to test whether ***sechash*** detects an incorrect -first argument, i.e., one outside the LBA range of the disk.

C11.12.2 Test setup

Any disk could be used.

C11.12.3 Test case dependencies

None.

C11.12.4 Procedure

Run ***sechash*** using a -first value larger than the address of the last sector on the disk:

```
sechash.csh shs-12 mcmillan serban /dev/sda CC –before -new_log -first 71687370 -last 71687380
```

Examine the standard output and/or the log file, if created, looking for an error message.

C11.12.5 Expected results

sechash issues an error message.

C11.13 shs-13

C11.13.1 Purpose

The purpose of this test case is to test whether ***sechash*** detects a -last argument outside the LBA range of the disk.

C11.13.2 Test setup

Any setup could be used.

C11.13.3 Test case dependencies

None.

C11.13.4 Procedure

Run ***sechash*** using a correct -first value, but a -last value larger than the address of the last sector on the disk:

```
sechash.csh shs-13 mcmillan serban /dev/sda CC –before
-new_log -first 71687300 -last 71687370
```

Examine the standard output and/or the log file, if created, looking for an error message.

C11.13.5 Expected results

sechash issues an error message.

C11.14 shs-14

C11.14.1 Purpose

The purpose of this test case is to test feature 5. Specifically, we want to test whether ***sechash*** displays its usage mode when requested by the *-h* option.

C11.14.2 Test setup

You may use any setup.

C11.14.3 Test case dependencies

None.

C11.14.4 Procedure

Run ***sechash*** without arguments; with incorrect arguments; with the *-h* option alone on the command line; and with the option *-h* with correct arguments. Capture the standard output:

```
sechash.csh > output.txt
sechash.csh shs-14 mcmillan serban -h >> output.txt
sechash.csh -h >> output.txt
sechash.csh shs-14 mcmillan serban /dev/sda CC -before -
first 7300 -last 7380 -h >> output.txt
```

C11.14.5 Expected results

sechash displays its usage mode in each case.

C12 *diskhash* Test Case Specifications

C12.1 dhs-01

C12.1.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and the selected hard disk drive. Specifically, we want to test whether ***diskhash*** creates a new log file with the default name for the -before option when no log file is present, correctly logs the hard disk drive, a one-word comment entered on the command line, and the program execution. Also, we want to test whether ***diskhash*** correctly computes the SHA-1 hash of the entire hard disk drive and logs the result.

C12.1.2 Test setup

Select and insert the target disk drive into a slot of the test computer. For example, assume that the target disk is the one externally labeled "CC" and it will be the Linux device /dev/sda. Boot up to Linux. Use the script cal-drive.csh available from NIST's "Computer Forensic Reference Data Set" (CFReDS) project to write a certain pattern on device /dev/sda whose SHA-1 and MD5 hashes are known. Invoke the cal-drive.csh script as follows:

```
cal-drive.csh sda
```

Save the log file cal-log.txt written by the script for later comparisons with the hashes computed by ***diskhash***.

C12.1.3 Test case dependencies

None.

C12.1.4 Procedure

Delete all log files from the log disk (the Linux boot disk).

Run ***diskhash*** using a one-word comment and the -before option:

```
diskhash.csh dhs-01 mcmillan serban /dev/sda CC -before -
comment HashDisk -hash sha1sum
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Compare the SHA-1 hash computed by ***diskhash*** to the value computed by the cal-drive.csh script.

C12.1.5 Expected results

diskhash creates a log file with the name "hashblog.txt" on the log disk.
diskhash logs the comment and the correct information required by features 1, 2, 4, 5.

C12.2 dhs-02

C12.2.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and selected hard disk drive. Specifically, we want to test whether ***diskhash*** appends the log records to an existing log file, correctly logs a multi-word comment entered on the command line, and the program execution. Also, we want to test whether ***diskhash*** correctly computes the MD5 hash of the selected disk drive.

C12.2.2 Test setup

Use the setup of test case dhs-01. Do not delete the log file created by test case dhs-01.

C12.2.3 Test case dependencies

dhs-01.

C12.2.4 Procedure

Run ***diskhash*** using the -*comment* option with a multi-word comment, the -before option, and select the MD5 function:

```
diskhash.csh dhs-02 mcmillan serban /dev/sda CC -before -
comment “Test MD5 hash” -hash md5sum
```

Use the *ls* command and a text editor to examine the log file’s existence, name, and contents. Compare the MD5 hash computed by ***diskhash*** to the value computed by the cal-drive.csh script.

C12.2.5 Expected results

diskhash appends the log records to the log file “hashblog.txt” created in the preceding test case.
diskhash logs the comment and the correct information required by features 1, 2, 4, and 5, computes the correct MD5 hash and logs it.

C12.3 dhs-03

C12.3.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and selected hard disk drive. Specifically, we want to test whether ***diskhash*** creates a new log file even though a log file with the same name already exists, prompts the user for a comment, logs that comment, correctly computes the SHA-1 hash of the hard disk drive, logs the result and program execution.

C12.3.2 Test setup

Use the setup of test case dhs-02.

C12.3.3 Test case dependencies

dhs-02. Do not delete the log file created or appended in that previous test case.

C12.3.4 Procedure

Run this test case right after dhs-02.

Run ***diskhash*** without the -*comment* option, but with -*new_log* and -before options.

```
diskhash.csh dhs-03 mcmillan serban /dev/sda CC -new_log
-before -hash sha1sum
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Compare the SHA-1 hash computed by ***diskhash*** to the value computed by the cal-drive.csh script.

C12.3.5 Expected results

diskhash creates a new log file "hashblog.txt", prompts the user for a comment, computes the correct SHA-1 hash, and logs the required information in the log file.

C12.4 dhs-04

C12.4.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and selected hard disk drive. Specifically, we want to test whether ***diskhash*** correctly computes the SHA-1 hash of the same hard disk drive as in the previous case after some changes to the disk contents, and creates a log file with a name appropriate for the -after option.

C12.4.2 Test setup

Use the setup of case dhs-01.

C12.4.3 Test case dependencies

None.

C12.4.4 Procedure

Use the ***diskchg*** tool to change a single byte somewhere within the test hard disk dive, for example the last byte of the last sector. Run ***diskhash*** using the -after option and specifying the same hard disk drive as in dhs-01. Do not use the -*hash* option, in order to test for the default hash computed:

```
diskhash.csh dhs-04 mcmillan serban /dev/sda CC -new_log
-after
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Compare the hash value logged by this case to the hash value computed in case dhs-01 or dhs-03.

C12.4.5 Expected results

diskhash creates a new log file "hashalog.txt", prompts the user for a comment, computes the correct SHA-1 hash, and logs the required information in the log file. The hash value recorded in hashalog.txt should be different from the one recorded in the previous cases.

C12.5 dhs-05

C12.5.1 Purpose

The purpose of this test case is to test features 1, 2, 3, 4, and 5 on the target computer and selected hard disk drive. Specifically, we want to test whether ***diskhash*** creates a log file with the name specified in the *-log_name* option, and correctly computes the MD5 hash of the entire disk.

C12.5.2 Test setup

Use the setup of case dhs-01.

C12.5.3 Test case dependencies

None.

C12.5.4 Procedure

Run ***diskhash*** using the *-log_name* option:

```
diskhash.csh dhs-05 mcmillan serban /dev/sda CC -log_name
diskhashlog.txt -hash md5sum
```

Use the *ls* command and a text editor to examine the log file's existence, name, and contents. Compare the hash value logged by this case to the hash value computed in case dhs-02.

C12.5.5 Expected results

diskhash creates a new log file of name "diskhashlog.txt".
diskhash prompts the user for a comment, which is correctly logged.
diskhash computes the MD5 hash of the entire disk, which should be different from the one computed in test case dsh-02.
seccmp logs the correct information required by features 2, 4, 5.

C12.6 dhs-06

C12.6.1 Purpose

The purpose of this test case is to test feature 5. Specifically, we want to test whether ***diskhash*** displays its usage mode when requested by the *-h* option.

C12.6.2 Test setup

Use the setup of dhs-01.

C12.6.3 Test case dependencies

None.

C12.6.4 Procedure

Run ***diskhash*** without arguments, with incorrect arguments, with the *-h* option alone on the command line, and with correct arguments plus the *-h* option. Capture its standard output:

```
diskhash.csh > output.txt
diskhash.csh dhs-06 mcmillan serban -logname -h >> output.txt
diskhash.csh -h >> output.txt
diskhash.csh dhs-06 mcmillan serban /dev/sda CC -before -h >> output.txt
```

C12.6.5 Expected results

diskhash displays its usage mode in each case.