NISTIR 7297-C

FS-TST 2.0: Forensic Software

Testing Support Tools

Code Review Report

January 2006

Paul E. Black
Information Technology Laboratory
National Institute of Standards And Technology
Gaithersburg, MD 20899

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.

Part A, *Test Plan, Test Design Specifications, and Test Case Specification*, is a companion document. 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[1] environment; however, some tests will require interaction with the MS-DOS operating system.

Part B, *Test Summary Report*, is an additional 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.

This document is Part C, *Code Review Report*. 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: Code review; computer forensic tool; disk imaging; software testing; testing support tools; FS-TST.

Acknowledgement

The following people participated in the code reviews in addition to the author: Eric Dalci (ED), James R. Lyle (JRL), Serban Gavrila (SG), Steve Mead (SM), and Kelsey Rider (KR).

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

Table of Contents

1 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. CFTT 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) is a software package that supports the testing of disk imaging tools. FS-TST 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.

These tools are required for testing disk imaging and software write-block tools. Being of general utility, these tools will be included in the test plans for deleted file recovery and other tools. This package of tools was originally written for the Microsoft MS-DOS operating system. The current version has been ported to compile and run on Unix systems, too. Since most of the tools need to deal with arcane details of the FAT file system, there was a complete code review.

This document covers the planning and specification of reviewing all the source code in the FS-TST package. It also reports the results of the code reviews.

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.

2 SCOPE

This covers tools to test programs that deal with hard disk drives at the application level. This does not deal directly with device drivers or other low-level hardware or software. Although some internal checks are made, the tools assume the presence of a valid FAT file system. The primary aim is to examine if the FS-TST tools correspond to their build specifications, rather than a comprehensive examination of whether the tools, used according to the test plan, adequately examine disk imaging tools. The secondary aim is to examine if the code is written so it is likely to reveal any latent errors in the code or any malformations in the file system. The code should err on the side of incorrectly reporting problems, instead of incorrectly reporting that all is well.

The expected operating environment is an MS-DOS or Unix operating system. The disk drives are FAT file systems.

In this Part C, the objects of examination are source code. This code is presumed to be the origin of the executable code whose testing is documented in Parts A and B. The compilation process is excluded. In plain English, we reviewed the source code for correctness, we tested the object code for behavior, and we presumed that the compilation process was faithful.

2.1 Items to Be Verified

This document uses the Computer Science sense of some terms. The verb "test" means to run a program. "Review" means to examine the source code of a program. "Validate" means to test or review to gain confidence that the program will behave as needed. Finally, "verify" means review to show that a program will satisfy its specification.[2]

There are several items to be reviewed. Each item is one source code file. All files, except zbios.c and zbios.h, correspond to executable programs with the same name. They can be grouped into four categories: programs to set something up, programs to document a test, programs to measure the result of a test, and other utilities and programs.

Set up

1. diskwipe.c
   Tag every sector of a hard disk with a unique pattern. Each sector has its location in the first 26 bytes[3]. Remaining bytes are filled with a selected value.
2. corrupt.c
   Change one byte of a hard disk.

Document

3. logcase.c
   Write information about the test run, e.g. tester, date, and disk names, to a log file.
4. logsetup.c
   Write details of the hard disk to a log file.
5. partab.c
   Write details of the partition tables of a file system to a log file.

Measure

6. diskcmp.c
   Compare two disks.
7. partcmp.c
   Compare two partitions.
8. adjcmp.c
   Compare two disks by partitions, allowing for cylinder alignment.

Utility

9. zbios.c
   Common utility routines to read a disk, traverse a partition, etc. This is a library, not a stand-alone program.
10. zbios.h
    Common macros and values. This is not a stand-alone program.

---

[2] Verification compares the program to what was requested: building the program right. Validation compares the program to what was needed: building the right program. For reasons ranging from mistakes to poor communication to changing needs, what was requested might not be the same as what is needed.

[3] Some older documentation states that the LBA is written into an 11-character field, and the 25th byte (byte number 24 counting from 0) is the null (end) byte. The current code writes a 12-character field (bytes 13-24), and the null byte is the 26th byte (byte number 25).

11. diskchg.c
    Change or examine the content of any sector.
12. seccmp.c
    Compare two sectors.

2.2 Summary of features to be verified

The features to be verified are explained in detail in FS-TST: Forensic Software Testing Support Tools: Requirements, Design Notes and User Manual [FST-RDU-20], particularly Sect. 2.

This section concisely describes what each program should do. Sect. 2.2.1 describes requirements for common functionality shared by most of the programs. Sect. 2.2.2 lists specific requirements for each program.

2.2.1 Common features

Most of the programs share common functionality. For example, most of them are required to log descriptive information about any disk drives they use. Rather than describe these common requirements with each program, they are described here once.

Every time a program that is required to log execution is executed, it must record names and versions of source files, when compiled, when executed, command line, test case identifier, etc. In addition, the program must have options to start a new log file or append to an existing one and to print a summary of command line operands and options.

Each hard disk drive used by a program is described in the program log file. A program required to log disks must record the type of BIOS access, the disk geometry, and, for IDE disk drives, other available information.

A program required to log partition tables must record starting and ending addresses, length, bootable flag, and partition type.

A program that compares a source disk, partition, or block of sectors to a similar destination assumes the source and destination were initialized by diskwipe. The program summarizes corresponding sectors (sectors compared, sectors matching, sectors differing, and the total number of bytes that differ) and records size information, if the source and destination are different sizes. If the destination is larger, it categorizes excess sectors as zero fill, diskwipe style fill, and other, and records the first few sectors of each category.

Except as noted, all programs must report I/O errors. They must also report invalid command line parameters or options.

2.2.2 Specific features

DISKWIPE

This program writes unique content to each disk sector. The first 26 (0-25) bytes of each sector is a string with both the C/H/S and LBA address of the sector. The remaining bytes are set to a specified fill byte value.

CORRUPT
Corrupt changes a single byte of an image file.

ADJCMP
Adjcmp compares two disks where the partitions copied to the destination are adjusted so that the copy is aligned on a cylinder boundary. It assigns each sector to a contiguous block of sectors, called a chunk, records the location of each chuck, categorizes each chunk, and allows the correspondence of source to destination chunks to be specified. For destination chunks without corresponding source chunks, it categorizes the sectors of the chunk as zero fill, diskwipe style fill, or other, and reports the first few sectors. It summarizes information about the number and type of boot tracks, partitions, unallocated chunks, and destination chunks without corresponding source chunks.

DISKCMP
Diskcmp compares two disks to evaluate the accuracy of a disk duplication operation.

PARTCMP
Partcmp compares two partitions to evaluate the accuracy of a partition duplication operation.

LOGCASE
Logcase creates a log file with basic information about the test case.

LOGSETUP
Logsetup logs information about the setup of a source disk.

PARTAB
Partab prints the partition table for a hard drive.

DISKCHG
Diskchg sets up a hard disk to help test other support programs. Diskchg can set a single byte to a given value, fill an entire sector with zero (0x00) bytes, or fill a sector in diskwipe style. Diskchg can also be used to examine the contents of a given sector.

SECCMP
Seccmp compares two disk sectors. If the sectors are not diskwipe style filled or zero filled, log any differences. Diskwipe style filled sectors or zero filled sectors are logged but not compared.

2.3 Transmittal of Items Reviewed

Two versions of code were reviewed. The older source code was reviewed first. The following transmittal is the newer, current code.

Some files were “DOS” style, in which lines end with a “carriage return” character (hexadecimal 0x0D), and some files were “Unix” style, in which lines did not end with a carriage return. Although this difference may not be visible, it affects the SHA1[1] value.

adjcmp.c Linux Version 1.4 Created 03/25/05
size: 36,600 bytes [sic]
SHA1: c0a9a8570d96903d090e53c9dd9630c6ce29afeb
File has no “carriage return” characters

corrupt.c Linux Version 1.2 Created 02/18/05
size: 6142 bytes
SHA1: 1dfbd8ea8368e2d81dbed91efc11989ec558f567
File has no “carriage return” characters

diskchg.c Linux Version 1.4 Created 03/15/05
size: 22,665 bytes
SHA1: 6724716b1cd3cc8681d974fb892fc55cdd87b5c9
Every line ends with a “carriage return” character

diskcmp.c Linux Version 1.2 Created 02/18/05
size: 11,866 bytes
SHA1: 0d4cb943a2fda059907d1a74b7372ce20366b86b
Every line ends with a “carriage return” character

diskwipe.c Linux Version 1.4 Created 03/18/05
size: 8992 bytes
SHA1: 78062f811075853d783b0744400b4d975b1941bf
Every line ends with a “carriage return” character

logcase.c Linux Version 1.2 Created 02/18/05
size: 2909 bytes
SHA1: 7dbda9fa9fc7c31db06967528e7d0fda96638005
Every line ends with a “carriage return” character

logsetup.c Linux Version 1.2 Created 02/18/05
size: 2964 bytes
SHA1: 4d49b67feb49dd81d2df8e4aa09a8576d245c42b
Every line ends with a “carriage return” character

partab.c Linux Version 1.4 Created 03/21/05
size: 4909 bytes
SHA1: 548eeec28250347afe2fc96800df94d70dd8b635
Every line ends with a “carriage return” character

partcmp.c Linux Version 1.3 Created 03/15/05
size: 18,619 bytes
SHA1: 248702ba1c619ce4c60f3e3e12cd0a1cbee355f9
File has no “carriage return” characters

seccmp.c Linux Version 1.3 Created 03/18/05
size: 13,183 bytes
SHA1: d2f7d2aebf7b7ff624f7069ec04873c794206049
File has no “carriage return” characters

zbios.c Linux Version 1.5 Created 03/21/05

[1] The Secure Hash Algorithm (SHA1), developed by NIST, along with the NSA, for use with the Digital Signature Standard (DSS) is specified in the Secure Hash Standard [SHS].

size: 25,803 bytes
SHA1: 5e88a03d2212e3dea7320fc30adb6f44ab57a2d1
File has no “carriage return” characters

zbios.h Linux Version 1.1 Created 02/10/05
size: 6923 bytes
SHA1: 68b8d436a2fd41d539baa1c585c20c3168fcd108
File has no “carriage return” characters

Supplier: Dr. James R. Lyle
National Institute of Standards and Technology
By email 14 April 2005
Address: james.lyle@nist.gov
100 Bureau Dr.
Gaithersburg, MD 20899

3 ANOMALIES

The reviewers found no anomalies in the final code review.

4 VALIDATION METHODOLOGY

The 10 tools in the package constitute almost 4,500 lines of C code. Most of the tools need to deal with arcane details of the FAT file system to accomplish their functions. The FS-TST package was originally written for the Microsoft MS-DOS operating system. The current version has been ported to compile and run on Unix. The many changes to code dealing with details of the FAT file system handling, the user interface, and other functions of the tools motivated a complete retest of the package. To increase confidence in the reliability of the code, there was also a complete code review.

The package was initially tested and the code reviewed soon after the porting to Unix. Because of the many code changes motivated by the result of the tests and reviews, the entire package was retested and reviewed again.

The FS-TST tools have features to test behavior to great precision. The tools have additional analysis features to help track down unexpected behaviors in the test procedures or a tool under test. In particular, diskwipe tags every sector with a unique identifier based on its location (so-called “toe tagging”) and fills the rest of the sector with a specified value. This allows other tools in the package to check with high assurance that copied data ends up in the right place. If the data is not imaged as expected, such tagging helps track down what happened. A less precise procedure, like comparing the hash values of entire disks, would only establish that the contents are not as expected.

4.1 Assumptions

The code review assumes the libraries, compiler, operating system, hardware, etc. have no pernicious flaws and are well-behaved. Considering the general lack of significant or systematic flaws in those components and absence of errors found in the extensive executions in testing phase, documented in Parts A and B, the assumption is justified.

The tools assume there is no subtle bit shifting within a sector or bit substitution across sectors. Otherwise there may be a possibility that the sector “toe tagging” and filling scheme might not be sufficient to detect that data is not in the right place or changed.

4.2 Test and Review Approach

No testing methodology can hope to be complete for programs of more than minimal complexity. An extreme example would be a tool with a so-called “Trojan horse”, i.e. a piece of code to perform a very unexpected function, say, *incorrectly* image a disk *only* if the computer's date is Sunday, 6 April 2014. Code inspection can catch such problems.

Testing and review together are likely to find simple errors, but will they find complex, subtle, or non-localized errors? Researchers have found support [DLP Hints] [Offutt] [KWG Interact] for the “fault-coupling hypothesis”: tests for simple faults are likely to find complex faults, too. Thus, there is support that checking for simple errors suffices.

Generally each program was reviewed separately, from the more fundamental programs to the more independent. Diskwipe was reviewed first, since many of the other programs depend on sectors being tagged in “diskwipe style” and since it used basic zbios.c routines. Diskcmp was reviewed next, since it used additional zbios.c routines. Partab was third, because it was the simplest use of the complex partition table access code. Most reviewers attended these first reviews to learn about the code and to learn a consistent review style. Routines in zbios.c and code in zbios.h were reviewed with the first program that used them. Reviews were kept to about an hour or an hour and a half.

After the first, complete code review, many small changes were made to the code. Rather than completely review all the code again, we used the Unix `diff` program to compare the new code with the reviewed code and display lines that were changed. For the second code review, just these changes were examined and justified.

The third code review examined string null termination concerns noted in the second review. The programs use many string initializations like

```
char        log_name[NAME_LENGTH] = "cmplog.txt";
```

In traditional C, only the first 11 characters are initialized. In ANSI C, remaining characters are initialized to zero (null). This review assumes ANSI C.

5 Technical Background

Hard disk drives consist of one or more round, flat plates or disks. When there is more than one disk, the disks are stacked on the same axis with a little space between them. Information is stored on the surfaces of the disks. The information is written and read by *heads* while the disks turn under them. A set of heads, one for each surface, is moved in and out from the edge to center together. (Some disks have more than one set of heads, spaced around the circumference, so all data can be accessed in less than one rotation.)

One head corresponds to one surface of one disk. As a disk turns, a stationary head traces a ring-shaped region of the disk surface, called a *track*, which passes under it. The set of tracks for all the heads at a particular radius from the axis form a hollow *cylinder* of surfaces. Thus a cylinder corresponds to the distance of the heads from the axis of rotation. For operational convenience, a track is divided into arcs. Each arc is called a *sector*.

Any piece of data in the disk drive may be addressed by its cylinder (distance from the center), head (surface), and sector (location around the disk) numbers. The three coordinates, "cylinder, head, and sector", are sometimes shortened to c/h/s. The number of cylinders (possible discrete locations from the axis to the edge), heads (surfaces), and sectors (per track) is called the *geometry* of a disk drive.

As disks increased in size, the maximum number of sectors per track exceeded software capabilities. For these disks to work with older software, the disk drive electronics might report a geometry different than the physical geometry. For instance, it might report twice the number of heads and half the number of sectors per track. A request for an even numbered head was translated into the second half of a track.

As disk drive capacity grew even more, even those translations were inadequate. The electronics within disk drives also grew in complexity to include sophisticated capabilities, such as caching, prefetching, and buffering. These changes increasingly distort the relation between the c/h/s passed to a disk drive and how long the drive took to read or write a sector. Newer drives put more sectors in tracks near the edge, which are physically longer, than tracks closer to the center. Such differing numbers of sectors per track cannot even be represented directly in the c/h/s paradigm. Increasingly, disk drive capacity is simply reported as the total number of sectors, a *Logical Block Address* (LBA), and the sectors are accessed sequentially. All mapping between the sector number and the distance from the center (cylinder), surface (head), and position within the track (sector) is left to the drive itself.

Software in the operating system, interfaces, and file system design limit the reported sizes to 1024 cylinders, 16 heads, and 63 sectors. These maxima may be reported even when a larger, correct LBA sector number is reported.

6 References

[DITS] "Disk Imaging Tool Specification." Version 3.1.6, National Institute of Standards and Technology, October 2001.

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

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

[IEEE Std 829] IEEE Std 829-1998, "Standard for Software Test Documentation," Institute of Electronic and Electrical Engineers, 1998.

[FST-RDU-20] "FS-TST: Forensic Software Testing Support Tools. Requirements, Design Notes, and User Manual." Version 2.0, National Institute of Standards and Technology, September 2004.

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

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

[SHS] "Secure Hash Standard (SHS)", FIPS Publication 180, National Institute of Standards and Technology, May 1993.

7 Interpretation of Code Review Results

This section summarizes of the actual results of the code reviews. Because of a mistake in formatting the code for printing, long lines were wrapped, and lines numbers given here may be larger than the actual line number in the code.

7.1 Code Review Results Summary Key

Table 7-1 describes each section of a code review report summary.

Table 7-1 Description of Code Review Report Summary

Heading	Description
Code review number and name of software reviewed.	
Case Summary	Details of the code being reviewed.
Reviewers	Name or initials of person(s) reviewing the code.

Heading	Description
Review Date	Date(s) of the review.
Anomalies	This part has one subsection for each file with anomalies, that is, items that may be worthy of note. The file name is given at the beginning of each section. Each subsection has one or more notes. Each note has the relevant line number(s) and the motivation for the note. The relevant code may be included. There may be a clarify comment. The last line of each note is in *italic* and is the assessment of the impact on testing forensic tools. A blank line separates notes.
Review Highlights	This part has one subsection for each file reviewed. The file name is given at the beginning of each section. Each subsection has one or more notes. Each note has the relevant line number(s) and the motivation for the note. The relevant code may be included. There may be a clarify comment. A blank line separates notes.
Results	A determination of whether the code reviewed should satisfy its requirements.

7.2 Summaries of First Review

Code Review 1 – diskwipe and some related code from zbios	
Case Summary	Review diskwipe.c, zbios.h, and some functions in zbios.c needed for diskwipe, lines 333-506: log_open(), log_close(), log_disk(), and feedback() diskwipe.c no version number; modified Aug 27 14:33 2004 zbios.h Version 3.1 Created 10/11/01 at 12:40:24 zbios.c Version 3.2 created 08/26/03 at 15:36:03
Reviewers	PEB, ED, SG, SM, KR
Review Date	16 December 2004
Anomalies	diskwipe.c Line 161 improperly initialized variable: from *should not cause invalid result* Lines 162 & 163 parameter ("AA") doesn't match format (%x)

	should not cause invalid result Lines 268 & 269 incorrect message if some errors occurred *should not cause invalid result* zbios.c Line 426 cylinder format should be %05, not %04 *should not cause invalid result*
Review Highlights	diskwipe.c Lines 65 & 66 cryptic variable names: hpc and spt. Add comment. Line 68 variable set, but used only in unreachable code: b. Line 74 constant (32256) should be in zbios.h and should be derived, i.e., 512*63. Line 84 possible divide by 0 (npc). Lines 104 - 107 unreachable code: not_ok always 0. If the code is changed so it is reachable, b may not be initialized (line 105). Line 142 (and zbios.h line 64) length of name of drive should be a macro in zbios.h. Possible buffer overflows if users enter really long strings, e.g., lines 172, 190, 198, 238, and 242. User can set multiple log files (e.g. lines 177, 179, and 191) and there is only a warning (lines 225 - 227). Line 205 user may enter negative number of heads. Line 266 (or 271) main() should return status. zbios.h Line 70 field set, but not used: geometry_is_real. Lines 159-162 macros expected near the beginning of the file.

	zbios.c Line 344 for consistency, p[3] and p[4] should be commented. Line 354 typo in comment: ... then use the[n] name ... Lines 437 - 439 if the disk doesn't have 63 sectors, there will be big problems. Check should be more visible (or removed). "63" should be a macro in zbios.h. Line 441 use macros: DRIVE_IS_IDE or DRIVE_IS_SCSI. Line 494 possible divide by 0 (ns) if from == to.
Results	Diskwipe should satisfy its requirements

Code Review 2 – rest of diskwipe-related code from zbios	
Case Summary	Review rest of functions in zbios.c needed for diskwipe, lines 64-101: print_rw_error() and mysync(), lines 131-261: probe_serial_model(), lines 507-597: disk_write() and disk_read(), and lines 636-679: open_disk() zbios.c Version 3.2 created 08/26/03 at 15:36:03
Reviewers	PEB, ED, SG, JRL, SM, KR
Review Date	20 December 2004
Anomalies	zbios.c Line 673 probe_serial_model() failure not handled. In event of an error, field fd may be negative (e.g. line 156) which is "true". *should not cause invalid result* Line 675 drive format should be %s, not %x. *should not cause invalid result*
Review Highlights	zbios.c Line 67 Use perror() instead of print_rw_error(). Line 68/80 No switch default. Line 91 Use print_rw_error() instead of switch. EROFS not in print_rw_error() Line 149 Allocate separate, more local buffers. Lines 170-172 Use access macros from zbios.h for d.

	Lines 186-188 and 216-218 Use memset(). Line 204 Explain impact on testing, e.g., "could not get serial number". Lines 240-243 Use strncpy() instead of save/set null/strcpy/restore. Lines 254 and 257 Possible buffer overflow; use strncpy(). Lines 518 and 566 Repeated computation. Make C/H/S to LBA conversion a macro. Line 529 Formats should be %llu. Line 536 Manifest constant. Use macro for 63*512. Line 540 Typo. Start sentence with capital, i.e., "An ... Line 577 Missing code. Print C/H/S (like line 529). Lines 648-653 Unused variables, b, cylinders, heads, sectors, and ec. Line 673, and associated requirement #3 in [FST-RDU-20], Sect. 2.1.2, may not be needed in a Unix environment.
Results	Diskwipe should satisfy its requirements

Code Review 3 – diskcmp and related code from zbios	
Case Summary	Review diskcmp.c, zbios.h, and functions in zbios.c needed for diskcmp, lines 263-332: create_range_list(), add_to_range(), and print_range_list(), and lines 680-704: lba_to_chs() and read_lba() diskcmp.c no version number; modified Aug 27 14:34 2004 zbios.h Version 3.1 Created 10/11/01 at 12:40:24 zbios.c Version 3.2 created 08/26/03 at 15:36:03
Reviewers	PEB, ED, SG, JRL, SM, KR
Review Date	6 January 2005
Anomalies	diskcmp.c Line 72 fossil code (-log name) in help message *should not cause invalid result*

	Lines 140 & 142 sscanf failure not checked *should not cause invalid result* zbios.c Line 270 malloc failure not checked *should not cause invalid result*
Review Highlights	diskcmp.c Line 67, etc. options follow operands; not typical style Lines 79, 80, 117, & 119 hard coded string lengths (12 and 80) Possible buffer overflow in strcpy if user enters really long strings, e.g., lines 145, 146, 171, & 179. Use strncpy. Lines 84 & 85 disk_control_ptr type defined, but not used Line 93 &ff byte_diffs may overflow? Lines 148 - 158 old, commented out code. Remove. Line 243 &ff source read error masks destination read error Lines 260, 321, & 325 manifest constant (512). Should be in zbios.h. Lines 323, 328, & 332 manifest constant (30). Should be in zbios.h and commented. Line 326 manifest constant (480). Should be derived and commented. zbios.c Common code in lines 296-298 and 302-304. add_to_range assumes offsets are increasing. Add comment. Line 692 comment is misleading: read_lba reads an entire track. Change to "Read the track containing sector "lba" of ..." Line 701 Comment why sector is set to 1.

	Does read_lba really read the sector desired? (And not, say, sector 1of each track instead?) Yes. read_lba reads an entire track along with that sector, but then sets b to the area of the buffer that has the sector. Tests confirm that it reads the right sector. read_lba converts lba to CHS, then calls disk_read, which converts it, back to lba. It might be clearer to use CHS directly?
Results	Diskcmp should satisfy its requirements

Code Review 4 – partab and related code from zbios	
Case Summary	Review partab.c, zbios.h, and functions in zbios.c needed for partab, lines 705-907: get_sub_part(), get_partition_table(), and print_partition_table() partab.c Version 3.1 Created 10/11/01 at 12:40:24
Reviewers	PEB, ED, SG, SM
Review Date	12 January 2005
Anomalies	zbios.c Lines 103-130 function trim() unused - probably doesn't work *should not cause invalid result* Line 723 malloc() return not checked *should not cause invalid result* Lines 756 & 807 Is type 0x0F an extended partition? *should not cause invalid result* Lines 726-748 and 786-798 casts to off_t in one but not in other *should not cause invalid result*
Review Highlights	partab.c Lines 80, 89, & 90 hard coded string lengths (12 and 80) Possible buffer overflow in strcpy if user enters really long strings, e.g., lines 101, 115, & 120. Use strncpy. Line 81 comment what log_ means Line 86 disk_control_ptr type defined, but not used Lines 94 and 102-105 old, commented out code. Remove. Line 128 move closer to line 136, where status is used

	zbios.c Lines 722 & 781 common, manifest constants (510 and 0xAA55). Comment, make into macros or function (e.g., is_partition_table()). Lines 726-748 and 786-798 (unpack buffer to pte_rec) common code Line 785 assign to status unused. Check status or cast to void. Lines 755, 756, 806, 807 use pt[i].type instead of re-accessing buffer Line 723 poor variable name. If you're using or changing this code, you really must mind your p's and q's. Suggestion: rename 'q' to 'head' or 'listHead'. Line 853 poor variable name. Suggestion: rename 'j' to be 'partitionNo' or 'partNum'. Lines 864 & 884 redundant code: type_code != 'X' Remove. Lines 866-874 and 885-895 (print partition) common code
Results	Partab should satisfy its requirements

Code Review 5 – corrupt	
Case Summary	Review corrupt.c corrupt.c Version 3.1 Created 10/11/01 at 12:40:26
Reviewers	SM, JRL, SG
Review Date	21 January 2005
Anomalies	*No anomalies found*
Review Highlights	corrupt.c Line 104 potential overflow, if more than 79 character comment Lines 119/120 comment on which operand has the file name does not match the code (p[4] vs. p[3]) Lines 120 and 122 the error code from a failed open() is not printed Suggestion: use perror() or something

	Lines 160, etc. if an incorrect offset is entered (3535g), are there any checks? Answer: both the command line and the value used by the program are logged, so we can see it. Line 155 error printed to stdout. Should go to stderr?
Results	Corrupt should satisfy its requirements

Code Review 6 – logcase	
Case Summary	Review logcase.c logcase.c Version 3.1 Created 10/11/01 at 12:40:26
Reviewers	SM, JRL, SG
Review Date	21 January 2005
Anomalies	*No anomalies found*
Review Highlights	logcase.c Line 78 Code allows for missing (or null) media (p[6] length > 0), but help and number of operands check require it. Fossil code?
Results	Logcase should satisfy its requirements

Code Review 7 – logsetup	
Case Summary	Review logsetup.c logsetup.c Version 3.1 Created 10/11/01 at 12:40:26
Reviewers	SM, JRL, SG
Review Date	21 January 2005
Anomalies	*No anomalies found*
Review Highlights	logsetup.c No validation of user input (disk, host, operator, OS, etc.). No suggestion. Line 70 no check for fopen() failure. Lines 64 and 75 Program requires 5 or more operands, but only logs the first 5. Also documentation and usage statement (line 48) refer to exactly 4. Suggestion: change line 64 to "if (np != 4)" and remove option printing (line 75). Or change documentation, help message, and log all options.
Results	Logsetup should satisfy its requirements

Code Review 8 – partcmp	
Case Summary	Review partcmp.c partcmp.c Version 3.1 Created 10/11/01 at 12:40:25
Reviewers	PEB, ED, KR
Review Date	21 January 2005
Anomalies	partcmp.c Line 122 type 0x0F is an extended partition type *may compare the wrong partitions* Line 480 no check for error reading destination *may incorrectly classify excess destination sectors*
Review Highlights	partcmp.c Line 175 no check for scanf failure Also: second % is useless Lines 222, 223, and 280 hard coded string lengths (12 and 80) Possible buffer overflow in strcpy if user enters really long strings, e.g., lines 330, 337, 344, and 345. Use strncpy. Line 230 comment that "static" is needed to run in some environments Line 260 log_diffs cannot be set true Could be a problem if the code breaks, then someone enables it for diagnostics. Suggestion: remove log_diffs and all code that is dead because of it. At least add a comment. Line 274 z_r could be zf_r (zero-filled range list) for consistency Lines 299, 301, 320, and 321 no check for sscanf failure Lines 394 and 395 fill bytes not logged Line 338 report the unknown operand, to help the user Line 423 no distinction between source and destination error Line 428, etc. unclear whether 1 means success or failure Line 464 use %llu for dst_n

	Line 492 manifest constant (30). Suggestion: make it a macro in zbios.h. Lines 478-511 common code with diskcmp.
Results	Partcmp should satisfy its requirements

Code Review 9 – diskchg	
Case Summary	Review diskchg diskchg.c Version 3.1 Created 10/11/01 at 12:40:26
Reviewers	SM, JRL, SG
Review Date	25 January 2005
Anomalies	*No anomalies found*
Review Highlights	diskchg.c Lines 97-101 use perror() instead Lines 117-122 dead code: rw_err cannot be negative, possibly a fossil? Lines 235 and 240 Confusing code: decode_disk_addr() called twice. Suggestion: comment that the first call decodes WHICH sector on the disk is written and the second call decodes WHAT is to be written there (that is, the sector the content says it came from). Line 335, 376, 471 etc. possible overflow. use strncpy to fill the log_name Line 339 lba is an inappropriate variable name, as it is an offset Line 495 manifest constant (512). #Define it?
Results	Diskchg should satisfy its requirements

Code Review 10 – adjcmp	
Case Summary	Review adjcmp.c adjcmp.c Version 3.1 Created 10/11/01 at 12:40:24
Reviewers	PEB, ED, KR
Review Date	25 & 27 January 2005
Anomalies	adjcmp.c Line 655 possible overflow: min should be off_t and initialize to

	MAX_OFF_T *may give incorrect results for very big disks* Lines 734 & 758 possible array overrun *may give incorrect results for disks with more than about 24 regions*
Review Highlights	adjcmp.c Duplicate code (lines 147-261): refactor scan_region() and use in diskcmp.c and partcmp.c. Line 194 dead code should report read error before exiting Duplicate code (lines 296-369): refactor and use in diskcmp.c and partcmp.c. Lines 319 and 326 error returned, but not used (line 572) Line 319 source read error masks destination read error Line 328, etc. manifest constants 512, 30, 480 Line 398 error returned, but not used (line 998) Lines 409, etc. poor variable names, suggestions: nm - matchingSectors uml - dml (destination match list) ml - sml (source match list) num - unmatchedDestRegions Lines 415 and 416 possible array overrun. Manifest constant (50). Is it related to manifest constant (25) in lines 808 and 816? Line 433 misspelled "chunk" Line 445 if source has fewer regions, dest 0 printed several times Suggestion: note which source sectors are not matched (-2?). Related: line 497 no way to say source doesn't match any dest Line 454 indicate that default is No Lines 455, 477, and 497 possible array overflow use %10s Lines 465 and 496 confusing prompt. Suggestion:

	Enter matching destination region (or -1 to list) Line 478 scanf()/while(){...scanf()} cleaner as for(;;){scanf(); if ... break; ...} Lines 477 and 497 possible array overrun: no check for valid destination region Lines 480 and 482 confusing output. Suggestion: Assigned/unassigned or Free/matched Line 504 confusing output. Say that it is reassigned. Lines 617-625 fossil code Line 647 misleading comment: "Find the lowest partition ..." Lines 664 and 684 use shared code to check if partition is extended Lines 686 - 690 does this work for 3(?) or more extended partitions? yes Line 826 fossil code Lines 838 & 840 no check for scanf failure of fill characters Lines 855 & 856 fill characters should be logged Lines 842, 843, 868, and 876 possible buffer overflow Lines 844 - 854 fossil code Line 877 print invalid operand so user knows which it is Line 883 unneeded code Line 881 return 1 in case of an error (e.g., line 867) Line 902 partition numbers should be logged not useful Lines 907 and 958 lseek or read failure is reported as no partition table

	Line 916 use %llu instead of %ld (and widen fields?)
Results	Adjcmp should satisfy its requirements

Code Review 11 – seccmp	
Case Summary	Review seccmp.c seccmp.c Version 3.1 Created 10/11/01 at 12:40:25
Reviewers	PEB, ED, KR
Review Date	28 January 2005
Anomalies	seccmp.c Lines 149, 150, 156, and 157 printing sector buffer with %s format *may print garbage if sector not initialized diskwipe style*
Review Highlights	seccmp.c Lines 49 - 207 Share scan_region() or other code from adjcmp.c instead of having it in-line here. Line 88 source read error masks dest read error Lines 105, 106, 109, etc. *different* manifest constant (26) Lines 105/118 and 106/138 different "characteristic" bytes Lines 127 - 131 just loop through whole sector to judge if zero-filled Line 172 separator too short. Code should be 8+... Lines 234, 237, 245, and 246 static string buffers Lines 259, 260, 286, and 293 possible buffer overflow Line 250 add comment: "needed for DOS environment" Line 253 should be np < 7 reviewer error - code is correct Command line operands 5 and 7 (src and dst fill) are never used Lines 261 - 268 fossil code Lines 302 and 303 no check for scanf failure

	Line 313 return 1 to indicate error Line 315 not needed Lines 321 and 325 no check for open_disk failure Line 323 time stamp in wrong place? Lines 339 and 344 improve prompt: "... or ^D to exit" Line 340 no check for scanf failure
Results	Seccmp should satisfy its requirements

7.3 Summaries of Second and Third Reviews – Current Code

The third review assumes ANSI C.

Code Review 1 – adjcmp	
Case Summary	Review adjcmp.c Linux Version 1.4 Created 03/25/05 against Version 3.1 created 10/11/01
Reviewers	PEB
Review Dates	29 April 2005 and 19 January 2006
Anomalies	*No anomalies found*
Review Highlights	line 426 possible buffer overflow scanf ("%s", ans); lines 435, 448, 853, 861, 902, 910 binary mega- is Mi, so BMB may be better as MiB (see http://physics.nist.gov/cuu/ Units/binary.html) printf ("%2d %c from %llu to %llu len=%llu %8.2fMB %8.2fBMB\n", lines 778 and 780 target of %x should be unsigned int, not int sscanf (p[5],"%2x",&is_fill);
Results	Adjcmp should satisfy its requirements

Code Review 2 – corrupt	
Case Summary	Review corrupt.c Linux Version 1.2 Created 02/18/05 against Version 3.1 created 10/11/01
Reviewers	PEB

Review Dates	29 April 2005 and 19 January 2006
Anomalies	*No anomalies found*
Review Highlights	*No problems found*
Results	Corrupt should satisfy its requirements

Code Review 3 – diskchg	
Case Summary	Review diskchg.c Linux Version 1.4 Created 03/15/05 against Version 3.1 created 10/11/01
Reviewers	PEB
Review Dates	29 April 2005 and 19 January 2006
Anomalies	*No anomalies found*
Review Highlights	lines 401 and 419 target of %x should be unsigned int, not int sscanf (p[i],"%x",&new_char_in);
Results	Diskchg should satisfy its requirements

Code Review 4 – diskcmp	
Case Summary	Review diskcmp.c Linux Version 1.2 Created 02/18/05 against no version number in code; modified Aug 27 14:34 2004
Reviewers	PEB
Review Dates	29 April 2005 and 19 January 2006
Anomalies	*No anomalies found*
Review Highlights	lines 133 and 134 string might not be null terminated strncpy(src_drive, p[4], NAME_LENGTH - 1);
Results	Diskcmp should satisfy its requirements

Code Review 5 – diskwipe	
Case Summary	Review diskwipe.c Linux Version 1.4 Created 03/18/05 against no version number in code; modified Aug 27 14:33 2004
Reviewers	PEB
Review Dates	29 April 2005 and 19 January 2006
Anomalies	*No anomalies found*
Review Highlights	lines 40 - 42 comments are wrong bytes 0-13 C/H/S address of the sector byte 14 blank character bytes 15-24 LBA address of the sector should be

	bytes 0-11 C/H/S address of the sector byte 12 blank character bytes 13-24 LBA address of the sector vide sprintf ((char *)(&(d->buffer[sector-1][0])), "%05llu/%03llu/%02llu %012llu", cylinder, head, sector, s); line 214 possible buffer overflow scanf("%s", ans);
Results	Diskwipe should satisfy its requirements

Code Review 6 – logcase	
Case Summary	Review logcase.c Linux Version 1.2 Created 02/18/05 against Version 3.1 created 10/11/01
Reviewers	PEB
Review Date	29 April 2005
Anomalies	*No anomalies found*
Review Highlights	*No problems found*
Results	Logcase should satisfy its requirements

Code Review 7 – logsetup	
Case Summary	Review logsetup.c Linux Version 1.2 Created 02/18/05 against Version 3.1 created 10/11/01
Reviewers	PEB
Review Date	29 April 2005
Anomalies	*No anomalies found*
Review Highlights	*No problems found*
Results	Logsetup should satisfy its requirements

Code Review 8 – partab	
Case Summary	Review partab.c Linux Version 1.4 Created 03/21/05 against Version 3.1 Created 10/11/01
Reviewers	PEB
Review Dates	29 April 2005 and 19 January 2006
Anomalies	*No anomalies found*

Review Highlights	line 107 possible buffer overflow sprintf (log_name, “%s”,p[i]);
Results	Partab should satisfy its requirements

Code Review 9 – partcmp	
Case Summary	Review partcmp.c Linux Version 1.3 Created 03/15/05 against Version 3.1 Created 10/11/01
Reviewers	PEB
Review Dates	29 April 2005 and 19 January 2006
Anomalies	*No anomalies found*
Review Highlights	lines 275 and 277 target of %x should be unsigned int, not int sscanf (p[5],"%2x",&fill_char); The original coder says that lines 388 and 389 were returned to their original, working forms.
Results	Partcmp should satisfy its requirements

Code Review 10 – seccmp	
Case Summary	Review seccmp.c Linux Version 1.3 Created 03/18/05 against Version 3.1 Created 10/11/01
Reviewers	PEB
Review Dates	29 April 2005 and 19 January 2006
Anomalies	*No anomalies found*
Review Highlights	line 91 wrong code - fault if src read ok, but dst read error if (src_status || dst_status) return src_status;
Results	Seccmp should satisfy its requirements

Code Review 11 – zbios	
Case Summary	Review zbios.c Linux Version 1.5 Created 03/21/05 against Version 3.2 created 08/26/03
Reviewers	PEB
Review Date	29 April 2005
Anomalies	*No anomalies found*
Review Highlights	lines 679/680 and 720/721 manifest constants (use is_extended macro) if ((mbr->pe[i].type_code == 0x05) || (mbr->pe[i].type_code == 0x0F)

	The original coder says that the change from absolute byte offset to mbr struct access returned the code to its original, working form.
Results	Zbios should satisfy its requirements

Code Review 12 – zbios.h	
Case Summary	Review zbios.h Linux Version 1.1 Created 02/10/05 against Version 3.1 Created 10/11/01
Reviewers	PEB
Review Date	29 April 2005
Anomalies	*No anomalies found*
Review Highlights	line 56 slightly safer to put parentheses around operand, e.g., #define is_extended(t) (((t) == 0x05) || ((t) == 0x0F))
Results	Zbios.h should satisfy its requirements

Appendix A Error Checklist for Code Review

A.1 Acknowledgements

Ideas for checklist questions came from the following sources.

John T. Baldwin, *An Abbreviated C++ Code Inspection Checklist*, Oct 1992, www.literateprogramming.com/Baldwin-inspect.pdf, accessed 19 May 2004.
Jorge Rady de Almeida Jr., João Batista Camargo Jr., Bruno Abrantes Basseto, and Sérgio Miranda Paz, *Best Practices in Code Inspection for Safety-Critical Software*, IEEE Software, 20(3):56-63, May/June 2003.
Java Code Inspection Checklist, www.isys.uni-klu.ac.at/ISYS/Courses/03WS/sete/literatur/L06-1, accessed 20 May 2004.
John Noll, *Code Inspection Checklist*, Jan 2004, www.cse.scu.edu/~jnoll/286/projects/checklist.html, accessed 19 May 2004.

A.2 Error Checklist

Review Date _____ Reviewer _____

Name of Code (function or file) _____

1 Data & Variables

1.1 Possible uninitialized variable? *No* ☐

1.2 Possible off-by-1 error in array indexing? *No* ☐

1.3 Possible array access out of bounds (or buffer overflow)? *No* ☐

1.4 Can a string* not *be null-terminated? *No* ☐

2 Calls & Returns

2.1 Wrong parameter order or type across call or return? *No* ☐

2.2 Parameters don't match format in *printf() or *scanf()? *No* ☐

2.3 Returned structures on stack? *No* ☐

2.4 Error return from function not checked? *No* ☐

3 Control Flow

3.1 Switch case without break (or return)? *No* ☐

3.2 Switch without default? *No* ☐

3.3 Possible infinite loop? *No* ☐

3.4 Incorrect comparison or Boolean operators (eg & vs. &&)? *No* ☐

4 Files

4.1 Possible reuse of temporary or working files? *No* ☐

Describe the location and nature of possible errors.