Salisbury UNIVERSITY

Honors College

Honors Thesis

An Honors Thesis Titled

File Survival on USB Drive

Submitted in partial fulfillment of the requirements for the Honors Designation to the

Honors College

of

Salisbury University

in the Major Department of

Math & Computer Science

by

Jessica Truong

Date and Place of Oral Presentation: April 28, 2017: SUSRC

Signatures of Honors Thesis Committee

Mentor:	[illegible]	Dr. Donald Spickler
Reader 1:	[illegible]	Dr. Michael Bardzell
Reader 2:	[illegible]	Dr. Timothy Stock
Director:	[illegible]	JAMES J. BUSS
	Signature	Print

Abstract

Flash (or USB) drives are small, inexpensive, portable, store an immense amount of data and can connect to multiple types of devices, including computers, digital cameras, printers, scanners, and external hard drives. In addition, these devices are one of the most used forms of personal data transfer and storage. Smaller versions of the same technology are used in nearly all mobile devices, such as cell phones, digital cameras, and tablet computers. If a flash drive is lost or stolen, and not encrypted, any information on the drive is open to the public, including confidential information. Many people are not aware that when you delete a file from a flash drive, or any conventional computer hard drive, it is not permanently gone. In fact, the file is not removed at all, the device simply allows another device to write over the deleted file if a new file is being written to the device. In our study, we explored how long a deleted file from a flash drive will remain readable or partially readable, as we added and removed other files from the device. Using freeware tools, such as Forensic Toolkit (FTK) and Autospy, we can view all the contents of a flash device, including deleted and partially overwritten media. Using a target test file, we wrote the file to the flash device, deleted it, then randomly wrote and deleted files of various sizes to the drive to test how long the target file will remain unaltered or partially readable. We will discuss the results of this study and we hope to raise awareness of possible data recovery on flash devices and that the misuse of such devices can cause a personal security issue.

Introduction

How can one little bug have control over an entire system without an administrator knowing? Today, cyber security is one of the most important topics to be discussed. It is an increasing problem because the public is unaware of the hackers continuously trying to steal people's personal and non-personal information. Although there are numerous types of cybercrimes, the most common occurrences within cyberspace are information warfare and digital crimes. Technology has enhanced and expanded over time to the point where anything is possible. There have been many real-world issues pertaining to hackers obtaining confidential information. One example, is the infamous Stuxnet Worm that emerged in 2010.

The Stuxnet Worm "was designed and released by a government (there have been rumors that it was the U.S. and Israel, however neither have confessed) to attack the Bushehr nuclear power plant in Iran" ("The Story Behind the Stuxnet Virus"). It was a "computer worm that targets industrial control systems that are used to monitor and control large scale industrial facilities like power plants, dams, waste processing systems, and similar operations" ("The Stuxnet Worm"). The spread of this virus was through a USB (Universal Serial Bus) drive. This was the first ever attack that allowed hackers to be able to exploit real-world equipment. This kind of attack demonstrates the importance of USB drives and how the public may not comprehend that a great deal can happen within a USB drive. It is critical to note that a USB drive should not be underestimated and that a flash drive can be a huge threat. Therefore, it is necessary for the general public to understand what a USB drive is and how it functions.

A USB (or flash, or thumb) drive is a small piece of equipment used to store and transfer data. It is used every day by people, companies, et cetera to perform many different tasks. Data is always being stored, transferred, or deleted. The specific file's location can be difficult to find because the file can be separated into several pieces, each having their specified location. It can be to the point where users have completely lost track of the data. The rise of flash drives came in 2007 “or even much earlier, some of the USB memory sticks offered by retailers could store 2000 times the capacity of floppy disks” (Amankwah-Amoah 127). Floppy disks were the main storage devices that were first used to store and transfer data. They were “one of the most widely used storage devices in the last quarter of the twentieth century” (Amankwah-Amoah 121). However, as the size of files became increasingly larger, some of the floppy disks became useless. People needed a storage device that can hold a large amount of data. Therefore, the rise of USB drives caused the use of floppy disks to decline. USB drives “played a pivotal role in the reliance on the floppy as a key mechanism for data transfer” (Amankwah-Amoah 127). It is a device that people depend on and use it on a day-to-day basis.

USB drives are solid-state drives and are nonvolatile devices because they can preserve data even when there is no power connected. The data is “stored directly in silicon NAND flash memory devices” (Siewart and Dane 30). Solid State Drives (SSD) do not have moving parts such as “the rotating media and servo-actuated read/write heads” (Siewart and Dane 30) that are used to access data from Hard Disk Drives (HDD). The advantage of the moving parts “improves the durability in resisting physical shock or mechanical failure and increases performance density” (Siewart and Dane 30). SSDs are able to perform at a much faster rate than HDDs. The data stored in SSDs are in a “virtual memory map on the NAND flash” (Siewart and Dane 30) and obtaining the data is not difficult. All that needs to be done is to change the address of the file to execute the next read/write whereas HDDs require multiple steps before one can access the data. SSDs also use flash memory, meaning they can retain data when there is no power. Thus, it can be safely assumed that USB flash drives are the new and improved floppy disks. Although flash drives have become popular and seem to be the most effective way of storing and transferring data, they can lose data just as easy.

The expression "expect the unexpected" perfectly characterizes a flash drive. One wrong click and everything could be gone. Flash drives are extremely convenient but can also be a disappointment when it comes to file storage. It can be frustrating when files are accidentally deleted since most people assume that the data will not be recoverable. There are many ways to recover deleted files by using forensic tools, like Forensic Toolkit (FTK) and Autopsy. Some of the tools are free or open source, while others must be paid for before they can be used. These tools can easily create a clone of the flash drive (FTK) and then use a separate tool to secure all information that exists on the flash drive. Autopsy allows us to view a list of all the deleted files and extract any that we want. It is not guaranteed that all files will still be intact and open fine. Depending on the file that we want to extract and view will depend on whether it has been previously deleted; if so, the number of files that has been added and deleted since the file has been deleted will affect its lifespan.

Files are stored differently depending on the type of system. Before any file can be saved on a system or media, the media first must be partitioned and formatted. The format of the volume is determined by the specific file system and the operating system (Linux, Windows, Mac OS X). A file system "defines the way the files are named, stored, organized, and accessed on volumes" (Kent, et al 10). Two common file systems are the File Allocation Table (FAT) and New Technology File System (NTFS). FAT uses tables to organize the files and folders. For each folder, the "starting cluster and date-time stamps associated with each file" can be seen (Casey 514). Each entry in the table designates what each specific cluster is being used for. If the cluster contains a zero, this means that the cluster is free and available to use. The FAT "file systems do not record the last accessed time, but only the last accessed date" (Casey 515). NTFS is different from FAT because it stores "file system information in several system files including a Master File Table, supporting larger disks more efficiently (resulting in less wasted space), and providing file and folder level security using Access Control Lists, and more" (Casey 519). The NTFS stores the minimal amount of information; this means when the master file table is created only the essential elements are in the table. The table gradually grows as more files are added. NTFS is "designed with disaster recovery in mind, storing a copy of the $BOOT system file at both the beginning and end of the volume" (Casey 519). A major difference between FAT and NTFS is that NTFS is an ownership file system which means that not all details of the file system are regularly available. Files are stored this way so they can be quickly accessed if needed. Learning about how files are stored on a flash drive is important, but understanding the safety and lifespan of a file is just as significant.

Files stored on flash drives can be overwritten easily. An important aspect of data recovery is file carving. If data on a flash drive has been "deleted some time in the past and the files are partially overwritten, they will only exist in the area of the drive that is unallocated" (Daniel 201). The purpose of file carving is "to find the pieces of the deleted files in unallocated space and electronically tape them back together" (Daniel 201). This is like taping pieces of a paper back together that has been shredded. There is a possibility that not one hundred percent of the specific file will be recoverable from file carving. This is because the operating system “will eventually use the space on the hard drive for new files and overwrite all or part of any file that was previously deleted” (Daniel 203). Any type of file can be recovered from unallocated space.

A question that may be asked about data recovery is how much information can be retrieved from a specific file on a USB drive? Exactly what percentage of said file can be recovered? Although using freeware tools such as Autopsy and Forensic Toolkit (FTK) will help recover deleted files from the flash drive, this does not mean that all the information from all files on the drive will still be there. Over time, the space on the flash drive will be overwritten with new files added and slowly remove bits of the deleted files.

USB drives are extremely important and popular to carry around. Flash storage devices are used as external storage if a system runs out of disk space and can no longer be usable if the drive was corrupted or stolen. The results of this study will hopefully raise awareness of possible data recovery on flash devices and that the misuse of such devices can cause a personal security issue.

Materials

- Computer (DELL Laptop)
- Flash Drives (x4)
    - PNY USB Drives 16 GB
- Forensic Tools
    - Sleuthkit Autopsy
    - Forensic Toolkit (FTK)
- Vbindiff (Command Prompt)
- Specialized Codes (Both written in C and are cross platform)

- Write & Delete Files
- Corruption Percentages

Obtaining & Processing the Data

For each of the trials, I used Forensic Toolkit (FTK) and Autopsy to perform my data collection. I examined target files of different file size and file type every trial. I used a specialized code that Dr. Spickler and I wrote to add/delete random files onto the USB drive. The code allowed me to decide the lower and upper bound for the new added files. I chose how many files I wanted to add onto and delete from the USB drive. Once I completed with the addition & deletion of files, I used Forensic Toolkit to create a duplicate copy of the USB drive that would then let me import the data into Autopsy. Autopsy displays all the contents on the flash drive (including the deleted ones) and organizes the files based on the file type. Below is a detailed explanation of what was completed for me to acquire my results.

Procedure/Methodology

First, I used a code that randomly added and deleted files from a USB drive. I did this several times before creating an image of the USB using Forensic Toolkit (FTK). After I created the image, I imported said image into Autopsy, allowing me to view all contents on the flash drive. Below are the steps I took to create an image and how to import the image to view the contents of the flash drive.

A. Forensic ToolKit (FTK)

1. Plug in the USB Drive
2. Click on "AccessData FTK Imager"
3. Click on *file* → *Create Disk Image* → Click *Next* → Select the correct USB

4. Click *add* → click *next*
5. Fill in the correction information for "Evidence Item Information"
6. Click *next*
7. Under "Image Destination Folder" → Look for a folder named "Images" → Double click on the folder then look for the correct folder with the corresponding number ("USB##")
   a. Choose the folder to place the image of the USB in
8. Give the image an "Image filename (excluding extension) → click *finish*

B. Autopsy

1. Click on "Create New Case"
2. Enter in the required information under "Enter New Case Information"
   a. Case Name: (ex: TestUSB1)
   b. Base Directory: (ex: C:\Users\Jessie\Desktop\USB\Cases)
3. Click *Next*
4. Enter in the case number (ex: TestUSB1)
5. Click *Finish*
6. A window will pop up requiring you to complete a few more steps before one is able to view the contents on the flash drive
   a. Enter Data Source Information
      i. Select source type to add: Image File
      ii. Browse for an image file:
         - ➢ Click *Browse* → (USB → Image → USB##)
      iii. Click *Next*

b. Configure Ingest Modules

    i. Click *Next*

c. Add Data Source

    i. Click *Finish*

After I collected my data, I analyzed the target file to see if it was corrupted, partially corrupted, or not corrupted. The tool, Vbindiff, will tell me where the target file differs from its original file. Autopsy helped me determine the lifespan of the target file.

C. Vbindiff (Command Line Tool)

   a. Open a command prompt

   b. Go to where the vbindiff folder is located

   c. Then type "vbindiff file1.ext file2.ext" and hit Enter

      i. file1 and file2 are the same file – one is the original and the other is the one that was on the flash drive

   d. A split screen window will appear showing the two different files that are being compared

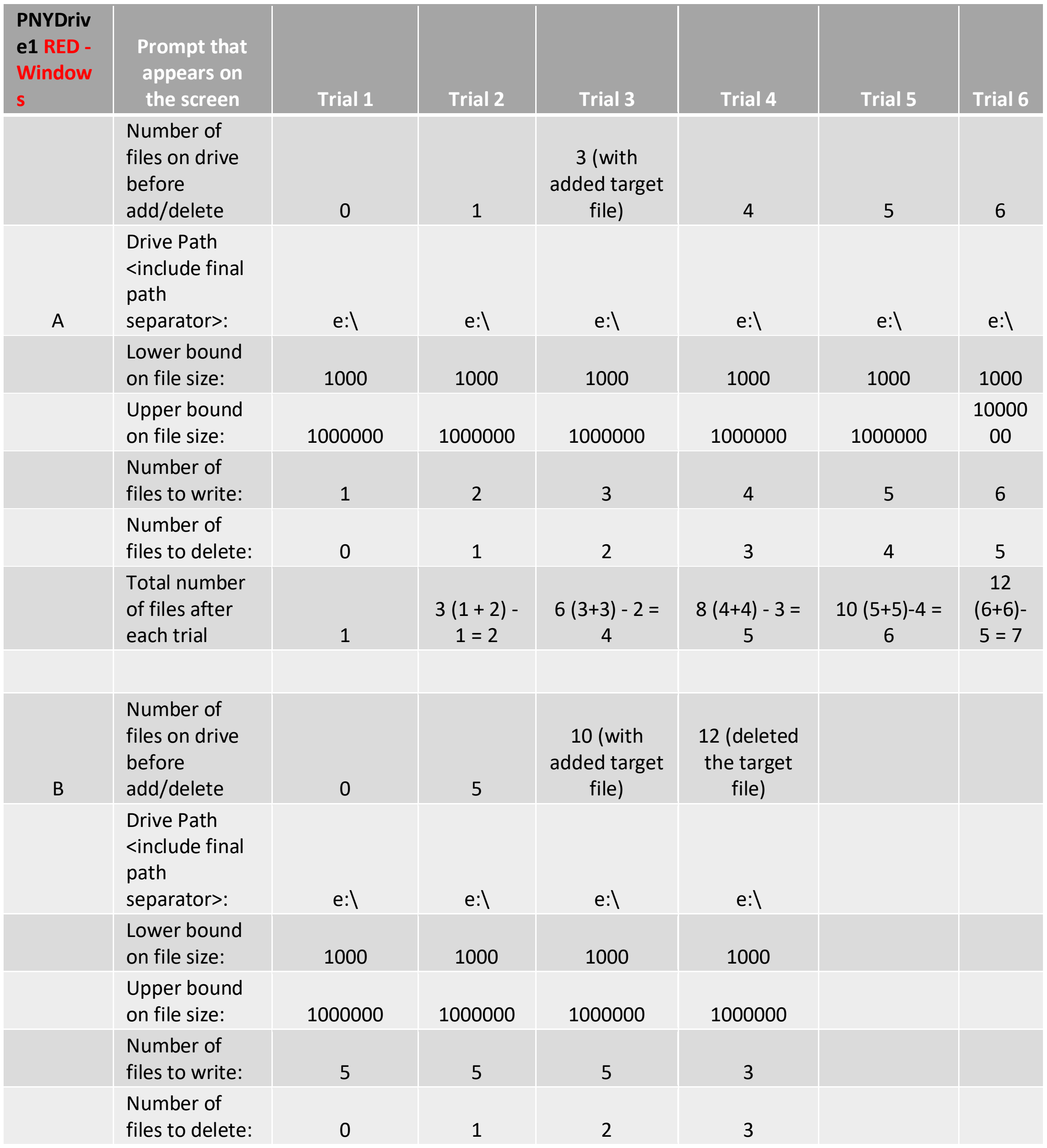
   e. The red portion represents the differences between the two files

Once I am finished with a specific run, I reformat the flash drive so that I can start the process all over again (erases information from the file allocation table). To do this, I right click on the USB drive and click format. A pop up window appears that says "WARNING: Formatting will erase ALL data on this disk. To format the disk, we click OK. To quit, click CANCEL." I then hit OK and then it will reformat the USB drive.

Data Collection

I used an Excel spreadsheet and Microsoft OneNote to record all my information. This helped me determine the best way to represent my data in a way that would make it easy for those unfamiliar of the topic to make sense of it. Below are screenshots from both Excel and Microsoft OneNote.

Excel:



PNYDrive1 RED - Windows	Prompt that appears on the screen	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6
	Number of files on drive before add/delete	0	1	3 (with added target file)	4	5	6
A	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\	e:\	e:\
	Lower bound on file size:	1000	1000	1000	1000	1000	1000
	Upper bound on file size:	1000000	1000000	1000000	1000000	1000000	1000000
	Number of files to write:	1	2	3	4	5	6
	Number of files to delete:	0	1	2	3	4	5
	Total number of files after each trial	1	3 (1 + 2) - 1 = 2	6 (3+3) - 2 = 4	8 (4+4) - 3 = 5	10 (5+5)-4 = 6	12 (6+6)-5 = 7
B	Number of files on drive before add/delete	0	5	10 (with added target file)	12 (deleted the target file)		
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\		
	Lower bound on file size:	1000	1000	1000	1000		
	Upper bound on file size:	1000000	1000000	1000000	1000000		
	Number of files to write:	5	5	5	3		
	Number of files to delete:	0	1	2	3		

	Total number of files after each trial	5	10 (5+5)-1 = 9	15(10+5)-2 = 13	15(12+3)-3 = 12		
C	Number of files on drive before add/delete	0	4 (with added target)	2 (deleted the target file)			
	Drive Path <include final path separator>:	e:\	e:\	e:\			
	Lower bound on file size:	1000	1000	1000			
	Upper bound on file size:	1000000	1000000	1000000			
	Number of files to write:	5	0	1			
	Number of files to delete:	2	1	0			
	Total number of files after each trial	3	4 - 1 = 3	3 (2+1)-0 = 3			
D	Number of files on drive before add/delete	1 (target file)					
	Drive Path <include final path separator>:	e:\					
	Lower bound on file size:	1000					
	Upper bound on file size:	1000000					
	Number of files to write:	4					
	Number of files to delete:	1					
	Total number of files after each trial	5-1 = 4					
E	Number of files on drive before add/delete	1 (target file)	5				
	Drive Path <include final path separator>:	e:\	e:\				

	Lower bound on file size:	1000	1000				
	Upper bound on file size:	1000000	1000000				
	Number of files to write:	4	0				
	Number of files to delete:	0	3				
	Total number of files after each trial	5	5-3 = 2				
F	Number of files on drive before add/delete	1 (target file)	5	5 (target file got deleted at trial 2)			
	Drive Path <include final path separator>:	e:\	e:\	e:\			
	Lower bound on file size:	1000	1000	1000			
	Upper bound on file size:	1000000	1000000	1000000			
	Number of files to write:	4	1	1			
	Number of files to delete:	0	1	2			
	Total number of files after each trial	5	6(5+1) -1 = 5	6(5+1)-2 = 4			
G	Number of files on drive before add/delete	1	2 (manually add in target file)				
	Drive Path <include final path separator>:	e:\	e:\				
	Lower bound on file size:	1000	1000				
	Upper bound on file size:	1000000	1000000				
	Number of files to write:	1	0				
	Number of files to delete:	0	2				
	Total number of files after each trial	1	0				

H	Number of files on drive before add/delete	0	5 (target file added)	4 (deleted target file before trial 3)			
	Drive Path <include final path separator>:	e:\	e:\	e:\			
	Lower bound on file size:	1000	1000	1000			
	Upper bound on file size:	1000000	1000000	1000000			
	Number of files to write:	4	1	0			
	Number of files to delete:	0	1	2			
	Total number of files after each trial	4	6(5+1) - 1 = 5	2			
I	Number of files on drive before add/delete	0	5 (manually add target file)	7	7 (target file got deleted after trial 3)		
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\		
	Lower bound on file size:	1000	1000	1000	1000		
	Upper bound on file size:	1000000	1000000	1000000	1000000		
	Number of files to write:	4	4	4	1		
	Number of files to delete:	0	2	4	1		
	Total number of files after each trial	4	9(5+4)-2 = 7	11(7+4)- 4 = 7	8(7+1) - 1 = 7		
J	Number of files on drive before add/delete	0	4	7 (manually add target file)	7	6 (manually delete target file)	9
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\	e:\	e:\
	Lower bound on file size:	1000	1000	1000	1000	1000	1000
	Upper bound on file size:	1000000	1000000	1000000	1000000	1000000	1000000
	Number of files to write:	4	4	4	1	3	0

	Number of files to delete:	0	2	4	1	0	2
	Total number of files after each trial	4	8(4+4)-2 = 6	11(7+4)-4 = 7	8(7+1) - 1 = 7	9(6+3)-0 = 9	9-2 = 7
K	Number of files on drive before add/delete	1 (target file)	5	3 (manually delete target file)			
	Drive Path <include final path separator>:	e:\	e:\	e:\			
	Lower bound on file size:	1000	1000	1000			
	Upper bound on file size:	1000000	1000000	2000000			
	Number of files to write:	4	2	1			
	Number of files to delete:	0	3	0			
	Total number of files after each trial	5	7(5+2)-3 = 4	4(3+1) - 0 = 4			
L	Number of files on drive before add/delete	1 (target file)	3				
	Drive Path <include final path separator>:	e:\	e:\				
	Lower bound on file size:	1000	1000				
	Upper bound on file size:	4000000	3000000				
	Number of files to write:	2	2				
	Number of files to delete:	0	1				
	Total number of files after each trial	(with the target file at the beginning) 3	5 (3+2) - 1 = 4				
M	Number of files on drive before add/delete	1 (target file)	6 (manually delete target file)				

	Drive Path <include final path separator>:	e:\	e:\				
	Lower bound on file size:	1000000	1000				
	Upper bound on file size:	4000000	2000000				
	Number of files to write:	6	2				
	Number of files to delete:	0	1				
	Total number of files after each trial	7	8(6+2) - 1 = 7				
N	Number of files on drive before add/delete	0	7 (manually add target file)	8 (target file got deleted by system)	8	10	
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\	e:\	
	Lower bound on file size:	1000	1000	1000	1000	1000	
	Upper bound on file size:	1000000	1000000	1000000	1000000	1000000	
	Number of files to write:	6	3	1	2	0	
	Number of files to delete:	0	2	1	0	3	
	Total number of files after each trial	6	10(7+3) - 2 = 8	9(8+1) - 1 = 8	10 (8+2) - 0 = 10	7 (10-3) = 7	
O	Number of files on drive before add/delete	1 (manually add target file)	5	6 (manually delete target file)			
	Drive Path <include final path separator>:	e:\	e:\	e:\			
	Lower bound on file size:	1000	1000	100			
	Upper bound on file size:	2000000	1000000	1000			
	Number of files to write:	4	4	1			
	Number of files to delete:	0	2	1			

	Total number of files after each trial	5	9 (5+4) - 2 = 7	7 (6+1) - 1 = 6			
P	Number of files on drive before add/delete	1 (manually add target file)	4 (manually delete target file)				
	Drive Path <include final path separator>:	e:\	e:\				
	Lower bound on file size:	1000	100				
	Upper bound on file size:	1000000	1000				
	Number of files to write:	4	2				
	Number of files to delete:	0	1				
	Total number of files after each trial	5	6 (4+2) - 1 = 5				
Q	Number of files on drive before add/delete	0	11 (manually add target file)	10 (manually delete target file)			
	Drive Path <include final path separator>:	e:\	e:\	e:\			
	Lower bound on file size:	100	100	100			
	Upper bound on file size:	1000000	1000	1000			
	Number of files to write:	10	1	1			
	Number of files to delete:	0	1	1			
	Total number of files after each trial	10	12 (11+1) - 1 = 11	11(10+1)- 1 = 10			
R	Number of files on drive before add/delete	0	11 (manually add target file)	10 (manually delete target file)	10	12	
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\	e:\	

	Lower bound on file size:	100	100	100	1000	1000	
	Upper bound on file size:	1000000	1000	1000	1000000	1000000	
	Number of files to write:	10	1	1	2	0	
	Number of files to delete:	0	1	1	0	2	
	Total number of files after each trial	10	12(11+1) - 1 = 11	11(10+1)- 1 = 10	12 (10+2) - 0 = 12	12 - 2 = 10	
S	Number of files on drive before add/delete	0	11 (manually add target file)	10 (manually delete target file)	10		
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\		
	Lower bound on file size:	100	100	100	1000		
	Upper bound on file size:	1000000	1000	1000	1000000		
	Number of files to write:	10	1	1	2		
	Number of files to delete:	0	1	1	0		
	Total number of files after each trial	10	12(11+1) - 1 = 11	11(10+1)- 1 = 10	12 (10+2) - 0 = 12		
T	Number of files on drive before add/delete	0	11 (manually add target file)	10 (manually delete target file)	10		
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\		
	Lower bound on file size:	100	100	100	1000		
	Upper bound on file size:	1000000	1000	1000	1000000		
	Number of files to write:	10	1	1	1		
	Number of files to delete:	0	1	1	0		
	Total number of files after each trial	10	12(11+1) - 1 = 11	11(10+1)- 1 = 10	11 (10+1) - 0 = 11		

U	Number of files on drive before add/delete	0	11 (manually add target file)	11 (manually delete target file)	12		
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\		
	Lower bound on file size:	100	100	100	1000		
	Upper bound on file size:	1000000	1000	1000	1000000		
	Number of files to write:	10	2	2	1		
	Number of files to delete:	0	1	1	0		
	Total number of files after each trial	10	13(11+2) - 1 = 12	13(11+2)- 1 = 12	13 (12+1) - 0 = 13		
V	Number of files on drive before add/delete	0	7 (manually add target file)	7(manually delete target file)	7	9	
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\	e:\	
	Lower bound on file size:	1000	1000	1000	1000	1000	
	Upper bound on file size:	1000000	1000000	1000000	1000000	1000000	
	Number of files to write:	6	3	1	2	0	
	Number of files to delete:	0	2	1	0	3	
	Total number of files after each trial	6	10(7+3) - 2 = 8	8(7+1) - 1 = 7	9 (7+2) - 0 = 9	9(9-3) = 6	
W	Number of files on drive before add/delete	0	6	10 (manually add target file)	8	8 (manually deleted target file)	
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\	e:\	
	Lower bound on file size:	1000	100	100	1000	2000	
	Upper bound on file size:	1000000	1000	1000	1000000	5000000	
	Number of files to write:	6	5	0	1	1	

	Number of files to delete:	0	2	2	0	2	
	Total number of files after each trial	6	11(6+5) - 2 = 9	10+0 - 2 = 8	8 + 1 - 0 = 9	9(8+1) - 2 = 7	
X	Number of files on drive before add/delete	0	8 (manually add target file)	9	5 (manually delete target file)		
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\		
	Lower bound on file size:	4000000	4000000	4000000	4000000		
	Upper bound on file size:	12000000	12000000	12000000	12000000		
	Number of files to write:	7	4	0	2		
	Number of files to delete:	0	3	3	0		
	Total number of files after each trial	7	12(8+4) - 3 = 9	9-3 = 6	5 + 2 = 7		
Y	Number of files on drive before add/delete	0	10	26 (manually add target file)	26	29 (manually delete target file)	31
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\	e:\	e:\
	Lower bound on file size:	4000000	4000000	4000000	4000000	4000000	4000000
	Upper bound on file size:	12000000	12000000	12000000	12000000	12000000	12000000
	Number of files to write:	10	20	5	12	4	10
	Number of files to delete:	0	5	5	8	2	5
	Total number of files after each trial	10	30 (10+20) - 5 = 25	31 (26+5) - 5 = 26	38 (26+12) - 8 = 30	33 (29+4) - 2 = 31	41 (31+10) - 5 = 36
Z	Number of files on drive before add/delete	1 (target file)	10	12 (target file got deleted by system)	18		
	Drive Path <include final	e:\	e:\	e:\	e:\		

	path separator>:						
	Lower bound on file size:	4000000	4000000	4000000	4000000		
	Upper bound on file size:	12000000	12000000	12000000	12000000		
	Number of files to write:	9	6	8	2		
	Number of files to delete:	0	4	2	3		
	Total number of files after each trial	10	16 (10+6) - 4 = 12	20 (12+8) - 2 = 18	20 (18+2) - 3 = 17		
PNYDrive2 RED - Windows							
AA	Number of files on drive before add/delete	1 (target file)	11	13	14 (target got deleted by the system)		
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\		
	Lower bound on file size:	4000000	1000	2000	4000000		
	Upper bound on file size:	12000000	1000000	4000000	12000000		
	Number of files to write:	10	5	6	2		
	Number of files to delete:	0	3	5	0		
	Total number of files after each trial	10	16 (11+5) - 3 = 13	19 (13+6) - 5 = 14	16 (14 + 2) - 0 = 16		
BB	Number of files on drive before add/delete	1 (target file)	11	18 (target file got deleted by system)			
	Drive Path <include final path separator>:	e:\	e:\	e:\			
	Lower bound on file size:	1000000	20000000	1000			
	Upper bound on file size:	4000000	35000000	1000000			
	Number of files to write:	10	10	1			
	Number of files to delete:	0	3	0			

	Total number of files after each trial	11	21(11+10) - 3 = 18	19 (18+1) - 0 = 19			
CC	Number of files on drive before add/delete	1 (target file)	9	17 (target file got deleted by sytstem)	19		
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\		
	Lower bound on file size:	20000000	20000000	25000000	25000000		
	Upper bound on file size:	35000000	35000000	35000000	35000000		
	Number of files to write:	8	10	4	1		
	Number of files to delete:	0	2	2	0		
	Total number of files after each trial	9	19 (9+10) - 2 = 17	21 (17 + 4) - 2 = 19	20 (19+1) - 0 = 20		
DD	Number of files on drive before add/delete	0	5	7 (manually add target file)	21 (manually delete target file)	23	
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\	e:\	
	Lower bound on file size:	20000000	20000000	20000000	20000000	20000000	
	Upper bound on file size:	35000000	35000000	35000000	35000000	35000000	
	Number of files to write:	5	1	20	10	1	
	Number of files to delete:	0	0	5	8	5	
	Total number of files after each trial	5	6 (5+1) - 0 = 6	27 (7 + 20) - 5 = 22	31 (21 + 10) - 8 = 23	24 (23+1) - 5 = 19	
EE	Number of files on drive before add/delete	0	10	12 (manually add target file)	13	15 (target file got deleted by system)	
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\	e:\	

	Lower bound on file size:	20000000	40000000	40000000	20000000	40000000	
	Upper bound on file size:	35000000	10000000 0	100000000	35000000	100000000	
	Number of files to write:	10	2	4	6	3	
	Number of files to delete:	0	1	3	4	2	
	Total number of files after each trial	10	12 (10+2) - 1 = 11	16 (12 + 4) - 3 = 13	19 (13 + 6) - 4 = 15	18 (15+3) - 2 = 16	
FF	Number of files on drive before add/delete	0	5 (manually delete file)	9	17 (target file got deleted by system)	23	
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\	e:\	
	Lower bound on file size:	40000000	20000000	40000000	40000000	40000000	
	Upper bound on file size:	80000000	35000000	80000000	80000000	80000000	
	Number of files to write:	4	6	10	8	10	
	Number of files to delete:	0	2	2	2	2	
	Total number of files after each trial	4	11(5+6) - 2 = 9	19 (9+10) - 2 =17	25 (17+8) - 2 = 23	33 (23+10) - 2 = 31	
GG	Number of files on drive before add/delete	1 (target file)	10	15 (target file deleted by system)	18		
	Drive Path <include final path separator>:	e:\	e:\	e:\	e:\		
	Lower bound on file size:	40000000	40000000	40000000	40000000		
	Upper bound on file size:	80000000	80000000	80000000	80000000		
	Number of files to write:	10	8	5	10		
	Number of files to delete:	1	3	2	5		
	Total number of files after each trial	10	18 (10+8) - 3 = 15	20 (15+5) - 2 = 18	28 (18+10) - 5 =23		

Microsoft OneNote:

- I used Microsoft OneNote to record the lower and upper bound of each trial, FTK Elapsed Time, the result obtained through Autopsy, and other notes

PNYRedDrive1 - WINDOWS

- Capacity of USB: 14.5 GB
- File System: FAT 32 (Default)
- Allocation Unit Size: 8192 bytes
- The size of the first target file is 891 bytes (.txt files)
    - Salisbury University Core Statement w/edits ("I love dogs" at the end)
- In this set of data collection the range for the lower bound is less than 1KB and the upper bound is equal to 1 MB

A	Lower bound: 1 KB Upper bound: 1 MB	FTK elapsed time: 0:10:29 Autopsy: the target file was completely corrupted after six trials, with adding and deleting with the same range for the lower and upper bound	• Used the same lower bound and upper bound for all six trials • At trial 4, the target file got automatically deleted (done by the system) • Total number of deleted files on USB = 15
B	Lower bound: 1 KB Upper bound: 1 MB	FTK elapsed time: 0:10:28 Autopsy: target file is completely corrupted	• Used the same lower bound and upper bound for all five trials • Manually deleted the target file before trial 4 • Total number of deleted files on USB = 7
C	Lower bound: 1 KB Upper bound: 1 MB	FTK elapsed time: 0:10:24 Autopsy: target file is completely corrupted	• Used the same lower bound and upper bound for all three trials • Manually added target file at trial 2 • Manually deleted target file before trial 3 • Total number of deleted files on USB = 4
D	Lower bound: 1 KB Upper bound: 1 MB	FTK elapsed time: 0:10:30 Autopsy: the content of the target file is the same as the original file - none of the file got corrupted (checked with vbindiff)	• Used the same lower bound and upper bound for one trial • Manually add the target file (placed it in my USB before adding/deleting files)
E	Lower bound: 1 KB	FTK elapsed time: 0:10:30 Autopsy: the content of the target file is the same as the	• Used the same lower bound and upper bound for two trials

	Upper bound: 1MB	original file - none of the file got corrupted (checked with vbindiff)	• Began with the target file on the USB • Total number of deleted files on USB = 3
F	Lower bound: 1 KB Upper bound: 1 MB	FTK elapsed time: 0:10:29 Autopsy: target file is completely corrupted - there were random letters that matched the original file (checked with vbindiff)	• Used the same lower bound and upper bound for three trials • Target file got deleted at trial 2 • Total number of deleted files on USB = 3
G	Lower bound: 1 KB Upper bound: 1 MB	FTK elapsed time: 0:10:30 Autopsy: the content of the target file is the same as the original - no corruption to it	• Used the same lower bound and upper bound for two trials • Manually add the target file at the beginning of trial 2 • Total number of deleted files on USB = 2
H	Lower bound: 1 KB Upper bound: 1 MB	FTK elapsed time: 0:10:30 Autopsy: the content of the target file is the same as the original - no corruption to it	• Used the same lower bound and upper bound for all three trials • Manually add the target file at the beginning of trial 2 • Manually deleted the target file at the beginning of trial 3 • Total number of deleted files on USB = 4

PNYRedDrive1 - WINDOWS

- The size of the target file is 103KB (.docx file)
  - Study Abroad Application
- In this set of data collection the range for the lower bound is less than 1KB and the upper bound is equal to 1 MB

I	Lower bound: 1KB Upper bound: 1MB	FTK elapsed time: 0:10:30 Autopsy: the content of the target file is the same as the original - no corruption to it; when I used vbindiff to compare the two files the words for both (original & target file) were in different characters	• Used the same lower bound and upper bound for all four trials • Manually add the target file at the beginning of trial 2 • Target file got deleted after trial 3 • Total number of deleted files on USB = 7 (8-carved files)
J	Lower bound: 1KB Upper bound: 1MB	FTK elapsed time: 0:10:27 Autopsy: target file is completely corrupted; some random letters match up in both files	• Used the same lower bound and upper bound for all six trials • Manually added target file at the beginning of trial 3

			• Manually deleted target file at the beginning of trial 5 • Total number of deleted files on USB = 10
K	**First two trials**: Lower bound: 1 KB Upper bound: 1 MB **Third Trial**: Lower bound: 1 KB Upper bound: 2MB	FTK elapsed time: 0:10:30 Autopsy: target file is completely corrupted; some random letters match up in both files	• Target file was already on the USB drive (to begin with) • Manually deleted the target file at the beginning of trial 3 • Total number of deleted files on USB = 4
L	**Trial 1** Lower bound: 1 KB Upper bound: 4 MB **Trial 2** Lower bound: 1 KB Upper bound: 3 MB	FTK elapsed time: 0:10:26 Autopsy: the content of the target file is the same as the original - no corruption to it	• Target file was already on the USB drive (to begin with) • At trial 2 the target file got deleted by the system • Total number of deleted files on USB = 1
M	**Trial 1** Lower bound: 1MB Upper bound: 4MB **Trial 2** Lower bound: 1KB Upper bound: 2MB	FTK elapsed time: 0:10:31 Autopsy: target file is completely corrupted; some random letters match up in both files • "The file StudyAbroad.docx cannot be opened because there are problems with the contents."	• Target file was already on the USB drive (to begin with) • Manually delete target file at Trial 2 • Total number of deleted files on USB = 2

PNYRedDrive - WINDOWS

- The size of the target file is 4.87 MB (.jpg)
  - House picture taken from archive.org (public domain images)
- The lower bound and upper bound is different for each run

N	Lower bound: 1KB Upper bound: 1MB	FTK elapsed time: 0:10:27 Autopsy: target file is completely corrupted • "Windows Photo Viewer can't open this picture	• Manually added target file (before trial 2) • Target file was deleted by the system after trial 2 and before trial 3

		because the file appears to be damaged, corrupted, or is too large"	• Total number of deleted files on USB = 6
O	**Trial 1** Lower bound: 1KB Upper bound: 2MB **Trial 2** Lower bound: 1KB Upper bound: 1MB **Trial 3** Lower bound: 100 bytes Upper bound: 1KB	FTK elapsed time: 0:10:30 Autopsy: target file is completely corrupted • "Windows Photo Viewer can't open this picture because the file appears to be damaged, corrupted, or is too large"	• Manually added target file (before trial 2) • Manually deleted target file before trial 3 • Total number of deleted files on USB = 4
P	**Trial 1** Lower bound: 1 KB Upper bound: 1 MB **Trial 2** Lower bound: 100 bytes Upper bound: 1 KB	FTK elapsed time: 0:10:28 Autopsy: target file is completely corrupted • "Windows Photo Viewer can't open this picture because the file appears to be damaged, corrupted, or is too large"	• Manually added target file (to begin with) • Manually deleted target file before trial 2 • Total number of deleted files on USB = 2
Q	**Trial 1** Lower bound: 100 bytes Upper bound: 1MB **Trial 2** Lower bound: 100 bytes Upper bound: 1KB **Trial 3** Lower bound: 100 bytes Upper bound: 1KB	FTK elapsed time: 0:10:29 Autopsy: the content of the target file is the same as the original - no corruption to it	• Manually added target file (before trial 2 began) • Manually deleted target file (before trial 3 began) • Total number of deleted files on USB = 3

R	**Trial 1** Lower bound: 100 bytes Upper bound: 1 MB **Trial 2** Lower bound: 100 bytes Upper bound: 1 KB **Trial 3** Lower bound: 100 bytes Upper bound: 1 KB **Trial 4** Lower bound: 1 KB Upper bound: 1 MB **Trial 5** Lower bound: 1 KB Upper bound: 1 MB	FTK elapsed time: 0:10:29 Autopsy: target file is completely corrupted • "Windows Photo Viewer can't open this picture because the file appears to be damaged, corrupted, or is too large"	• Manually added target file (before trial 2 began) • Manually deleted target file (before trial 3 began) • Total number of deleted files on USB = 5
S	**Trial 1** Lower bound: 100 bytes Upper bound: 1 MB **Trial 2** Lower bound: 100 bytes Upper bound: 1 KB **Trial 3** Lower bound: 100 bytes Upper bound: 1 KB **Trial 4** Lower bound: 1 KB Upper bound: 1 MB	FTK elapsed time: 0:10:22 Autopsy: target file is completely corrupted • "Windows Photo Viewer can't open this picture because the file appears to be damaged, corrupted, or is too large"	• Manually added target file (before trial 2) • Manually deleted target file (before trial 3 began) • Total number of deleted files on USB = 3 • When I add two random files with the random file size between 1KB and 1 MB the file is corrupted

T	**Trial 1** Lower bound: 100 bytes Upper bound: 1 MB **Trial 2** Lower bound: 100 bytes Upper bound: 1 KB **Trial 3** Lower bound: 100 bytes Upper bound: 1 KB **Trial 4** Lower bound: 1 KB Upper bound: 1 MB	FTK elapsed time: 0:10:30 Autopsy: the content of the target file is the same as the original - no corruption to it	• Manually added target file (before trial 2) • Manually deleted target file (before trial 3) • Total number of deleted files on USB = 3 • Performed the same number of trials like in Trial S except I am only adding in 1 file in trial 3
U	**Trial 1** Lower bound: 100 bytes Upper bound: 1 MB **Trial 2** Lower bound: 100 bytes Upper bound: 1 KB **Trial 3** Lower bound: 100 bytes Upper bound: 1 KB **Trial 4** Lower bound: 1 KB Upper bound: 1 MB	FTK elapsed time: 0:10:29 Autopsy: target file is completely corrupted • "Windows Photo Viewer can't open this picture because the file appears to be damaged, corrupted, or is too large"	• Manually added target file (before beginning trial 2) • Manually deleted target file (before beginning trial 3) • In both trial 2 and 3 instead of adding 1 random file with a size between 100 bytes to 1KB - I added two files with the same range for the lower and upper bound

PNYRedDrive1 - WINDOWS

- The size of the target file is 5.65 MB (.jpg)
    - Beach picture taken from creativecommons.org (public domain images)
- The lower bound and upper bound is different for each run

V	Lower bound: 1 KB	FTK elapsed time: 0:10:30	• Manually added the target file (before beginning trial 2)

	Upper bound: 1 MB	Autopsy: target file is completely corrupted • "Windows Photo Viewer can't open this picture because the file appears to be damaged, corrupted, or is too large"	• Manually deleted the target file (before the beginning trial 3) • Used the same upper and lower bound for all 5 trials
W	**Trial 1** Lower bound: 1 KB Upper bound: 1 MB **Trial 2** Lower bound: 100 bytes Upper bound: 1 KB **Trial 3** Lower bound: 100 bytes Upper bound: 1 KB **Trial 4** Lower bound: 1 KB Upper bound: 1 MB **Trial 5** Lower bound: 2 MB Upper bound: 5 MB	FTK elapsed time: 0:10:30 Autopsy: target file is completely corrupted • "Windows Photo Viewer can't open this picture because the file appears to be damaged, corrupted, or is too large"	• Manually add target file (before beginning trial 3) • Manually delete target file (before beginning trial 5) • Upper and lower bounds varied for each trial
X	Lower bound: 4 MB Upper bound: 12 MB	FTK elapsed time: 0:10:30 Autopsy: the content of the target file is the same as the original - no corruption to it	• Manually add target file (before beginning trial 2) • Manually deleted target file (after trial 3) • Upper and lower bound is the same for all four files
Y	Lower bound: 4 MB Upper bound: 12 MB	FTK elapsed time: 0:10:30 Autopsy: target file is completely corrupted • "Windows Photo Viewer can't open this picture because the file appears to	• Manually add target file (before beginning trial 3) • Manually delete target file (before beginning trial 5) • Upper and lower bound is the same for all files

		be damaged, corrupted, or is too large"	
Z	Lower bound: 4 MB Upper bound: 12 MB	FTK elapsed time: 0:10:25 Autopsy: target file is completely corrupted • "Windows Photo Viewer can't open this picture because the file appears to be damaged, corrupted, or is too large"	• Manually add the target file (placed it in my USB before adding/deleting files) • Target file got deleted by system after trial 2 • Upper and lower bound is the same for all four files
AA	**Trial 1** Lower bound: 4 MB Upper bound: 12 MB **Trial 2** Lower bound: 1 KB Upper bound: 1 MB **Trial 3** Lower bound: 2 KB Upper bound: 4 MB **Trial 4** Lower bound: 4 MB Upper bound: 12 MB	FTK elapsed time: 0:10:31 Autopsy: target file is completely corrupted • "Windows Photo Viewer can't open this picture because the file appears to be damaged, corrupted, or is too large"	• Manually added target file (placed it in my USB before adding/deleting files) • Target file got deleted after trial 3

PNYRedDrive2 - WINDOWS

- The size of the target file is 265 MB (.mov)
    - Video from vimeo (taken from creativecommons.org)
- Lower and upper bound is different for each trial

BB	**Trial 1** Lower bound: 1 MB Upper bound: 4 MB **Trial 2** Lower bound: 20 MB Upper bound: 35 MB **Trial 3**	FTK elapsed time: 0:10:31 Autopsy: the content of the target file is the same as the original - no corruption to it	• Manually added target file (placed it in my USB before adding/deleting files) • Target file got deleted by system after trial 3

	Lower bound: 1 KB Upper bound: 1 MB		
CC	Lower bound: 20 MB Upper bound: 35 MB	FTK elapsed time: 0:10:30 Autopsy: the content of the target file is the same as the original - no corruption to it	• Manually added target file (placed it in my USB before adding/deleting files) • Target file got deleted by system before trial 3
DD	Lower bound: 20 MB Upper bound: 35 MB	FTK elapsed time: 0:10:30 Autopsy: the content of the target file is the same as the original - no corruption to it	• Manually added target file (before trial 3) • Manually deleted target file (after target 3) • Used the same lower and upper bound
DD2	Lower bound: 20 MB Upper bound: 35 MB	FTK elapsed time: 0:10:29 Autopsy: I was able to retrieve the potus_target.mov file and a screenshot of the beginning of the video. I was not able to retrieve any of the random added/deleted files that had also been on the flash drive	• This trial was used to see if I could see any data on the flash drive (files from DD) even though I used "Quick Format"
EE	**Trial 1 & 4** Lower bound: 20 MB Upper bound: 35 MB **Trial 2, 3, 5** Lower bound: 40MB Upper bound: 100 MB	FTK elapsed time: 0:10:29 Autopsy: target file is completely corrupted • "Windows Media Player cannot play the file. The Player might not support the file type or might not support the codec that was used to compress the file"	• When I tried to use 50 MB - 1 GB the code freaked out and crashed

PNYRedDrive2 - WINDOWS

- The size of the target file is 26.9 MB (.mp4)
  - Video from vimeo (taken from creativecommons.org)
- Lower and upper bound is different for each trial

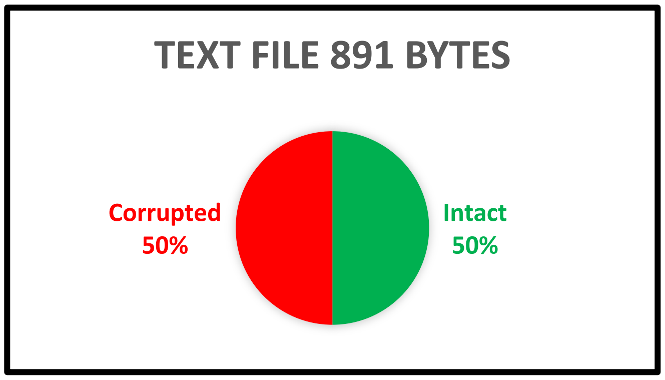
FF	**Trial 1, 3, 4, 5** Lower bound: 40 MB Upper bound: 80 MB **Trial 2**	FTK elapsed time: 0:10:30 Autopsy: the content of the target file is the same as the original - no corruption to it	• Manually added target file after trial 1 and before beginning trial 2 • Target file got deleted by system after trial 3 and before trial 4

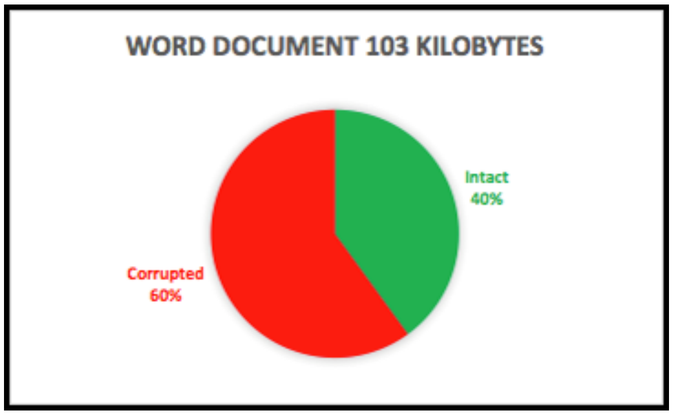
	Lower bound: 20 MB Upper bond: 35 MB		
GG	Lower bound: 40 MB Upper bound: 80 MB	FTK elapsed time: 0:10:33 Autopsy: the content of the target file is the same as the original - no corruption to it	• Manually added target file (placed it in my USB before adding/deleting files) • Target file got deleted by the system • Lower bund and upper bound are the same for all 4trials
HH	**Trial 2, 3, 5** Lower bound: 40 MB Upper bound: 80 MB **Trial 1 & 4** Lower bound: 20 MB Upper bound: 35 MB	FTK elapsed time: 0:10:31 Autopsy: the content of the target file is the same as the original - no corruption to it	• Manually add target file (before trial 3 & after trial 2) • Manually delete target file (before trial 4 began)
II	Lower bound: 40 MB Upper bound: 100 MB	FTK elapsed time: 0:10:33 Autopsy: the content of the target file is the same as the original - no corruption to it	• Manually added target file (placed it in my USB before adding/deleting files) • Target file got deleted by the system (before trial 3 and after trial 2)

Results & Analysis

I created pie charts to represent my data and each pie chart corresponds to a specific file. The chart shows the percentage of the specific file being destroyed and the percentage of the file intact. Each of the target files were taken from personal documents and non-profit websites.

- The number of files randomly added/deleted and the range for the randomly added file size will affect whether the target file is corrupted or not


TEXT FILE 891 BYTES
Corrupted 50%
Intact 50%


WORD DOCUMENT 103 KILOBYTES
Intact 40%
Corrupted 60%

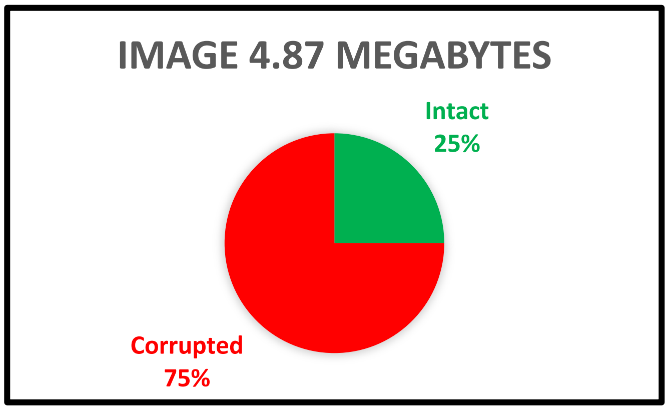


IMAGE 4.87 MEGABYTES
Intact 25%
Corrupted 75%

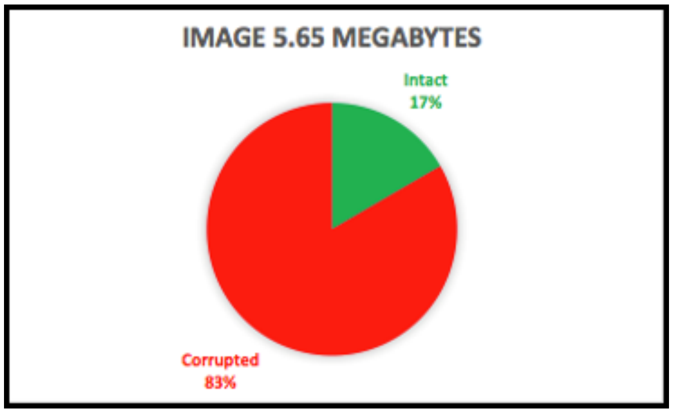
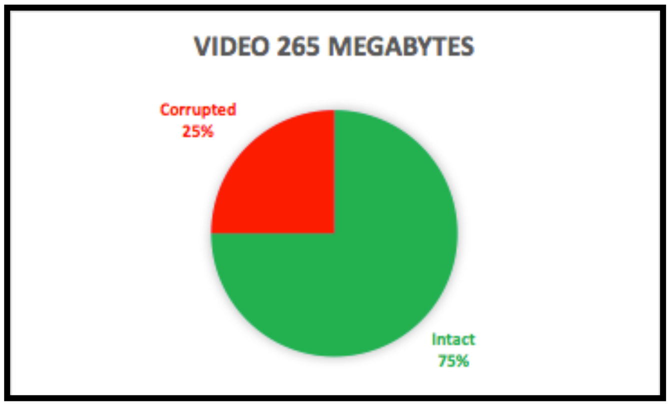
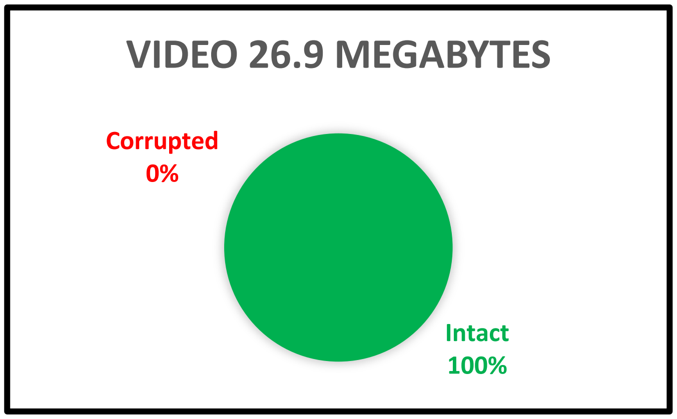


IMAGE 5.65 MEGABYTES
Intact 17%
Corrupted 83%





This table displays the type and size of the target file I used as well with my conclusion of what happened to each.

Target File	**Size**	**Results**
Text File	891 Bytes	• The same range for the randomly added file size was the same for all eight trials • Half of the time the file was corrupted and the other half the file was still intact
Word Document	103 Kilobytes	• The range for the randomly added files was between 1KB and 1 MB
Image	4.87 Megabytes	• More than half of the time the file is corrupted

		• If the range for the randomly added file is between 100 bytes and 1 MB then the file will be intact • As the size of the range for the random files increase, there is a greater chance that the file will be corrupted
Image	5.65 Megabytes	• The range for the randomly added file (depending on the trial) was between 1 MB and 35 MB
Video	265 Megabytes	• The range for the randomly added file (depending on the trial) was between 100 bytes and 12 MB • Using the same lower and upper bound range, adding and deleting files between 1-10 did not cause any damage to the file • However, adding over 10 files causes the file to be completely corrupted • In the last trial, when I tried to use 50 MB-1 GB as the range for the randomly added files – the code crashed
Video	26.9 Megabytes	• The range for the randomly added files was between 40 MB and 100 MB

		• Throughout all four trials the content of the videos were unharmed

Conclusion

The lifespan of files on a USB drive depends on the operating system the drive is connected to. From the data I have collected, I learned that the lifespan of files on USB drives on a Windows system is very slim. It does not take much for a target file to be completely corrupted once you delete it and add/delete more files to the flash drive. A factor to consider about file survival is how the file is saved on the system. Files can be saved in many ways on different operating systems. A point to note is the file systems that are on each of the operating system. Each of the file systems have unique features that distinguish themselves from one another. Because of this, the file's chance of surviving may be different depending on the operating system and file system.

Throughout this entire process, I have had a series of obstacles to overcome. The data collection process takes roughly ten minutes per trial so I was constantly running Autopsy and FTK. During my data collection, I noticed that there was a trend of being able to either retrieve all the information on a flash drive or no information. Since there was no possibility of recovering just part of a file, there was no need to use the corruption percentage code that Dr. Spickler and I created.

I learned an immense amount about USB drives and their importance. I also learned that the public is not well-informed about importance of USB drives and how you should be careful with the kind of information is being stored on there. From personal observations, when I explain how files are recoverable on USB drives it shocks people.

Future Directions

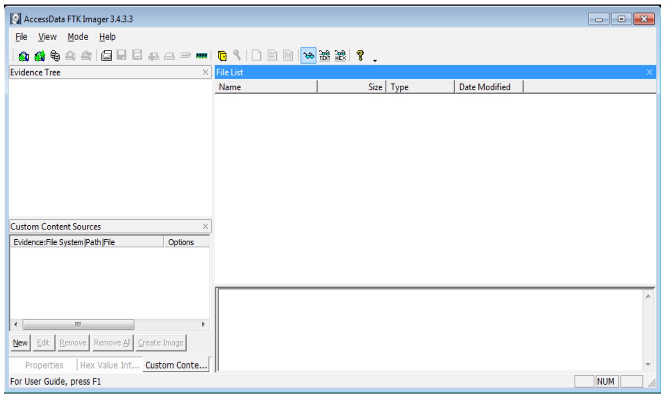
I would like to further my research by looking at different operating systems such as MAC, Linux (ext2, ext3, ext4). The difference in methodology of how files are saved in Linux might affect the lifespan of file survival.

Appendix

I. Forensic Toolkit (FTK)

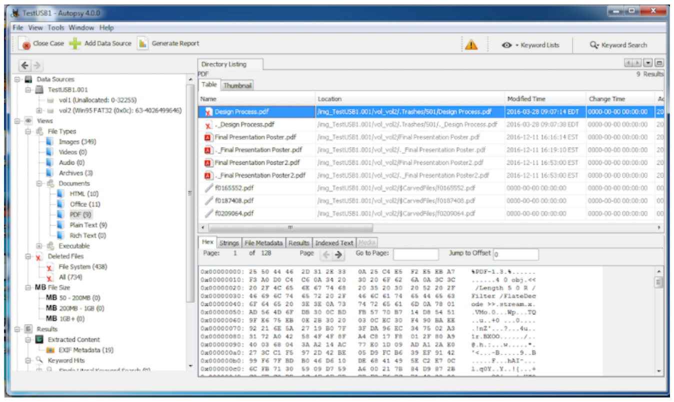
a. Creates a duplicate copy (image) of the USB drive

i. This window will appear when you first open FTK


AccessData FTK Imager 3.4.3.3
File View Mode Help
Evidence Tree
File List
Name Size Type Date Modified
Custom Content Sources
Evidence:File System|Path|File Options
New Edit Remove Remove All Create Image
Properties Hex Value Int... Custom Conte...
For User Guide, press F1
NUM

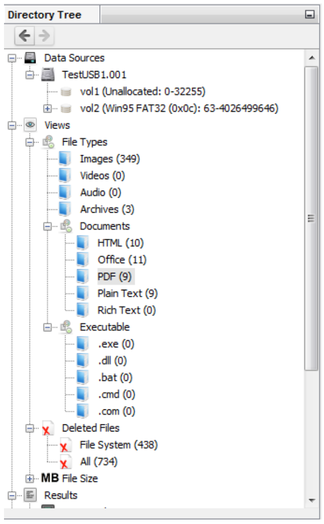
II. Sleuthkit Autopsy

a. Allows one to view all the contents on a flash drive (including the deleted ones)


TestUSB1 - Autopsy 4.0.0
File View Tools Window Help
Close Case Add Data Source Generate Report
Keyword Lists Keyword Search
Directory Listing
PDF
9 Results
Table Thumbnail
Hex Strings File Metadata Results Indexed Text Media
Page: 1 of 128 Page Go to Page: Jump to Offset 0

Name	Location	Modified Time	Change Time
Design Process.pdf	/img_TestUSB1.001/vol_vol2/.Trashes/501/Design Process.pdf	2016-03-28 09:07:14 EDT	0000-00-00 00:00:00
._Design Process.pdf	/img_TestUSB1.001/vol_vol2/.Trashes/501/._Design Process.pdf	2016-03-28 09:07:38 EDT	0000-00-00 00:00:00
Final Presentation Poster.pdf	/img_TestUSB1.001/vol_vol2/Final Presentation Poster.pdf	2016-12-11 16:16:14 EST	0000-00-00 00:00:00
._Final Presentation Poster.pdf	/img_TestUSB1.001/vol_vol2/._Final Presentation Poster.pdf	2016-12-11 16:19:10 EST	0000-00-00 00:00:00
Final Presentation Poster2.pdf	/img_TestUSB1.001/vol_vol2/Final Presentation Poster2.pdf	2016-12-11 16:53:00 EST	0000-00-00 00:00:00
._Final Presentation Poster2.pdf	/img_TestUSB1.001/vol_vol2/._Final Presentation Poster2.pdf	2016-12-11 16:53:00 EST	0000-00-00 00:00:00
f0165552.pdf	/img_TestUSB1.001/vol_vol2/$CarvedFiles/f0165552.pdf	0000-00-00 00:00:00	0000-00-00 00:00:00
f0187408.pdf	/img_TestUSB1.001/vol_vol2/$CarvedFiles/f0187408.pdf	0000-00-00 00:00:00	0000-00-00 00:00:00
f0209064.pdf	/img_TestUSB1.001/vol_vol2/$CarvedFiles/f0209064.pdf	0000-00-00 00:00:00	0000-00-00 00:00:00

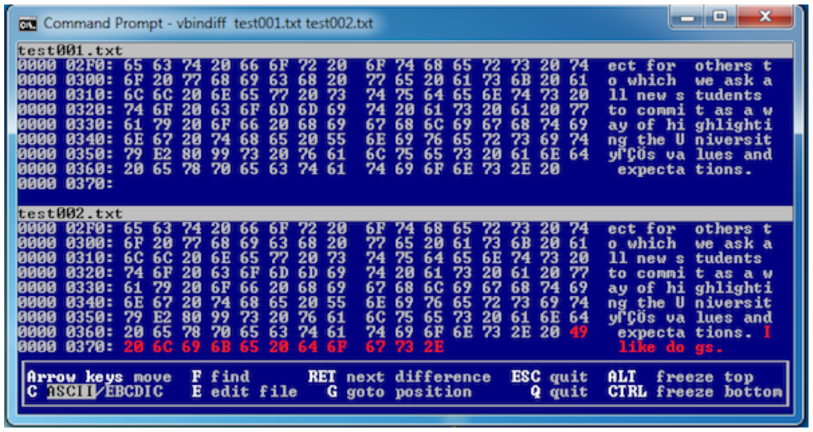
```
0x00000000: 25 50 44 46  2D 31 2E 33  0A 25 C4 E5  F2 E5 EB A7   %PDF-1.3.%......
0x00000010: F3 A0 D0 C4  C6 0A 34 20  30 20 6F 62  6A 0A 3C 3C   ......4 0 obj.<<
0x00000020: 20 2F 4C 65  6E 67 74 68  20 35 20 30  20 52 20 2F    /Length 5 0 R /
0x00000030: 46 69 6C 74  65 72 20 2F  46 6C 61 74  65 44 65 63   Filter /FlateDec
0x00000040: 6F 64 65 20  3E 3E 0A 73  74 72 65 61  6D 0A 78 01   ode >>.stream.x.
0x00000050: AD 56 4D 6F  DB 30 0C BD  FB 57 70 B7  14 D8 54 51   .VMo.0...Wp...TQ
0x00000060: 9F E6 75 EB  0E 2B 30 20  03 0C EC 30  F4 90 BA EE   ..u..+0 ...0....
0x00000070: 92 21 6E 5A  27 19 B0 7F  3F DA 96 EC  34 75 02 A3   .!nZ'...?...4u..
0x00000080: 31 72 A0 42  58 4F 4F 8F  A4 C8 17 F8  01 2F 80 A9   1r.BXOO....../..
0x00000090: 40 03 68 04  3A A2 14 AC  77 E0 1D 09  AD A1 2A E0   @.h.:...w.....*.
0x000000a0: 27 3C C1 F5  97 2D 42 BE  05 D9 FC B6  39 EF 91 42   '<...-B.....9..B
0x000000b0: 99 F6 7F BD  B0 46 D6 10  DE 68 41 49  5E C2 E7 0C   ......F...hAI^...
0x000000c0: 6C FB 71 30  59 09 D7 59  A6 00 21 7B  84 D9 87 2B   l.q0Y..Y..!{...+
```



```
Data Sources
  TestUSB1.001
    vol1 (Unallocated: 0-32255)
    vol2 (Win95 FAT32 (0x0c): 63-4026499646)
Views
  File Types
    Images (349)
    Videos (0)
    Audio (0)
    Archives (3)
    Documents
      HTML (10)
      Office (11)
      PDF (9)
      Plain Text (9)
      Rich Text (0)
    Executable
      .exe (0)
      .dll (0)
      .bat (0)
      .cmd (0)
      .com (0)
  Deleted Files
    File System (438)
    All (734)
  MB File Size
Results
```

III. Vbindiff

a. A command prompt tool that compares two files byte by byte and highlights the difference between the files in red



```
Command Prompt - vbindiff  test001.txt test002.txt
test001.txt
0000 02F0: 65 63 74 20 66 6F 72 20  6F 74 68 65 72 73 20 74  ect for  others t
0000 0300: 6F 20 77 68 69 63 68 20  77 65 20 61 73 6B 20 61  o which  we ask a
0000 0310: 6C 6C 20 6E 65 77 20 73  74 75 64 65 6E 74 73 20  ll new s tudents
0000 0320: 74 6F 20 63 6F 6D 6D 69  74 20 61 73 20 61 20 77  to commi t as a w
0000 0330: 61 79 20 6F 66 20 68 69  67 68 6C 69 67 68 74 69  ay of hi ghlighti
0000 0340: 6E 67 20 74 68 65 20 55  6E 69 76 65 72 73 69 74  ng the U niversit
0000 0350: 79 E2 80 99 73 20 76 61  6C 75 65 73 20 61 6E 64  yΓÇÖs va lues and
0000 0360: 20 65 78 70 65 63 74 61  74 69 6F 6E 73 2E 20      expecta tions.
0000 0370:

test002.txt
0000 02F0: 65 63 74 20 66 6F 72 20  6F 74 68 65 72 73 20 74  ect for  others t
0000 0300: 6F 20 77 68 69 63 68 20  77 65 20 61 73 6B 20 61  o which  we ask a
0000 0310: 6C 6C 20 6E 65 77 20 73  74 75 64 65 6E 74 73 20  ll new s tudents
0000 0320: 74 6F 20 63 6F 6D 6D 69  74 20 61 73 20 61 20 77  to commi t as a w
0000 0330: 61 79 20 6F 66 20 68 69  67 68 6C 69 67 68 74 69  ay of hi ghlighti
0000 0340: 6E 67 20 74 68 65 20 55  6E 69 76 65 72 73 69 74  ng the U niversit
0000 0350: 79 E2 80 99 73 20 76 61  6C 75 65 73 20 61 6E 64  yΓÇÖs va lues and
0000 0360: 20 65 78 70 65 63 74 61  74 69 6F 6E 73 2E 20 49   expecta tions. I
0000 0370: 20 6C 69 6B 65 20 64 6F  67 73 2E                   like do gs.

 Arrow keys move   F find        RET next difference   ESC quit   ALT  freeze top
 C ASCII/EBCDIC    E edit file     G goto position       Q quit   CTRL freeze bottom
```

This compares two files byte by byte and indicating any differences between the two files. One is the original file and the other has some changes in it

IV. Specialized Codes written by Dr. Spickler & I

a. Write and Delete Files

- Allows a user to add and delete as many files as they want
- The files that are deleted are done randomly by the system
- Users can decide on the size of files to be added to the USB drive

```
//This program adds/deletes random files to a USB drive
#include <fstream>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h> //format of directory entries
#include <string>
#include <ctime> // contains definitions of functions to get and
                    manipulate date & time information

using namespace std;

int NumFiles(string drive) //drive = path of specific  file
{
    int filecount = 0; //counter used to keep track of the number of
                            files

    DIR *dir; //DIR = a type representing a directory stream
    struct dirent *ent; //struct & pointer
    if ((dir = opendir (drive.c_str())) != NULL)   //opendir: likes
     character string
     //.c_str() converts a string into a character string
    {
        while ((ent = readdir(dir)) != NULL) //readdir: reads a
                                             //directory and returns
                                             //a pointer
            filecount++;
        closedir (dir);
    }
    else
    {
        return -1;
    }

    return filecount;
}
//As long as you can open the entry -> then you can read the
//directory and send all the information into the struct directory
//entry

//Read current directory -> all of the files currently on the flash
drive

//Prints out all the files in list form
//a. Set up directory
//b. Open directory
//c. Read directory -> sends information into struct ent
//d. Read in the file -> extract the file's name
void PrintDirList(string drive)
{
    DIR *dir;
    struct dirent *ent;
    if ((dir = opendir(drive.c_str())) != NULL)
    {
        while ((ent = readdir(dir)) != NULL)
            cout << ent->d_name << endl;

        closedir (dir);
    }
}

//*&dirlist = array of strings; pointer that it is an array
void GetDirList(string drive, string *&dirlist, int& filecount)
{
    filecount = NumFiles(drive); //gives you the number of files on
                                 //USB so you can create an array
                                 //that holds all files currently on
                                 //flash drive
    dirlist = new string[filecount];

    DIR *dir;
    struct dirent *ent;
    if ((dir = opendir(drive.c_str())) != NULL)//opens the directory
    {
      //read in each entry in USB and put entry's name into
      //directory list which is a string array
        for (int i = 0; i < filecount; i++)
        {
            ent = readdir(dir);
            dirlist[i] = ent->d_name;
        }
        closedir (dir);
    }
}

//Don't need this function
string RandomOneKBString()
{
    string retstr = "";

    for (long i = 0; i < 1024; i++)
    {
        char c = (char)(rand() % 256);
        retstr += c;
    }

    return retstr;
}

//File contents get written to the file
string RandomLengthString(long lower, long upper)
{
    string retstr = "";

    cout << "Creating String                              \r";
//overwrites the previous value -> writes on top of it; doesn't go
//to a new line

    float r = (float)rand()/RAND_MAX;
    long chars = lower + (long)(r*(upper - lower)); //creates a
                                                string between lower &
                                                upper in in it at
                                                random; I decide what
                                                the lower value and
                                                upper value is
    for (long i = 0; i < chars; i++)
    {
         //brings the value to ASCII region (rand() % 25))
         //char: random value; size of file; take enormous string
         //write to file
         //rand(): creates a random float
        char c = (char)(rand() % 256);
        retstr += c;
    }

    return retstr;
}
```

```
//Creates the random files
void WriteRandomFiles(string drive, int numfiles)
//parameters: bring in the  path where you want to write files and
//how many you want to create (this is determined in the main
//function
{
    cout << endl;

    for (int i = 0; i < numfiles; i++)
    {
        //allocates space for the newly created file
        char filename[100];

        //sprintf: formats the string
        sprintf(filename, "%sTestFile_%d%d%d.txt", drive.c_str(),
rand(), rand(), rand());

        //writing a string that is between 1000-10000 characters
        string writestr = RandomLengthString(1000, 10000);

        //what will be output to the screen
        cout << "Writing File: " << (i+1) << " of " << numfiles << "
\r";

        ofstream outfile(filename);
        outfile << writestr << std::endl;
        outfile.close();
    }
}

//*Cannot remove a directory*
void RemoveRandomFiles(string drive, long numfiles)
{
    string* dirlist = NULL;
    int filecount = 0;
    GetDirList(drive, dirlist, filecount);
    //drive: path; dirlist: pointer to string; filecount: # of files

    if (numfiles >= filecount)
    //if numfiles is greater than file count -> delete everything
    {
        for (int i = 0; i < filecount; i++)
        {

         //outputs to the screen the specific number of files being
         removed from the flash drive
            cout << "Removing File: " << (i+1) << " of " <<
filecount << "                \r";
            char filename[1000];
            sprintf(filename, "%s%s", drive.c_str(),
dirlist[i].c_str());
            remove(filename);
        }
    }
    else
    {
         //counter that keeps track of the number of files that have
         been //selected to be removed
        long filesselected = 0;
        int *check = new int[filecount];
        int jump = rand() % filecount;
        long pos = jump; //pos: position at which you will delete
                          //the file from; jump: first index we will
                          //delete

        for (int i = 0; i < numfiles; i++)
            check[i] = 0;

        while (filesselected < numfiles)
        {
         //if position equals 0 -> haven't selected the file yet;
         change value to 1 & updated file selected
         //if the index has not yet been marked at random to be
         deleted
         //decides at random which file we are going to delete at
         what index
            if (check[pos] == 0)
            {
                check[pos] = 1; //indicates we will delete file in
                                 specific position
                filesselected++;
            }
            pos += rand(); //pick a random position -> delete
                             specific file
            pos = pos % filecount; //if the position is greater than
                                    the number of files in the array
                                    then we need to do this so we can
                                    get a position in the array
        }

        long fileremovecount = 1;
        for (int i = 0; i < filecount; i++)
            if (check[i] == 1)
            {
                cout << "Removing File: " << fileremovecount << " of
" << numfiles << "                \r";
                fileremovecount++;
                char filename[1000];
                sprintf(filename, "%s%s", drive.c_str(),
dirlist[i].c_str());
                remove(filename);
            }
    }
}

int main()
{
   //srand: seeds the random number generator used by the function
   //rand: initializes random number generator
    srand(time(NULL));

    //the name of the USB (ex: "f:\")
    string drive;
    cout << "Drive Path (include final path separator): ";
    cin >> drive;

    //determine the number of files that are currently on the USB
    //drive
    int filecount = NumFiles(drive);

    cout << "\nNumber of Files: " << filecount << endl << endl;

    //prints out all of the files
    cout << "\nFile List: " << endl << endl;
    PrintDirList(drive);

    string* dirlist = NULL;
    filecount = 0;
    GetDirList(drive, dirlist, filecount);

    //prints out all of the files on the USB drive
    cout << "\nNumber of Files: " << filecount << endl << endl;

    for (int i = 0; i < filecount; i++)
        cout << dirlist[i] << endl;

    //accumlator of total number of bytes written
    WriteRandomFiles(drive, 100);

    //new number of files on USB drive
    filecount = NumFiles(drive);
    cout << "\nNumber of Files: " << filecount << endl << endl;

    RemoveRandomFiles(drive, 1); //deleting one file from USB drive

    //prints out the number of files on the drive after files have
     //been deleted
    filecount = NumFiles(drive);
    cout << "\nNumber of Files: " << filecount << endl << endl;

    //outputs all of the files (updated list without the deleted
    //files)
    PrintDirList(drive);

    return 0;
}
```

b. Corruption Percentages

- If the size of the original file is longer than the altered (target file) then the lost bytes would be considered as the differences and is added to the difference count
- The difference count is divided by the length of the original file

- If the size of the altered file is longer than the original file → considered the extra bytes as trash
- Take the difference count and divide it by the size of the original file

```
#include <fstream>
#include <iostream>

using namespace std;

std::ifstream::pos_type filesize(const char* filename)
{
    std::ifstream in(filename, ios::binary | ios::ate);
    return in.tellg();
}

int main()
{
    string file1; //creating a string named file1
    string file2; //creating a string named file2

    cout << "Filename #1 (Original): "; //Original file's name
    cin >> file1; //User input with the name of the original file
    cout << "Filename #2 (Test): "; //Target file's name (edits made on
                                  //original
    cin >> file2; //User input with the name of the target file

    ifstream inFile; //ifstream: File stream class used for file
                     //handling
    size_t size1 = 0; //size_t: Represent the size of any object in
                      //bytes
    size_t size2 = 0;

   //Opens the first file (original)
    inFile.open(file1.c_str(), ios::in|ios::binary);
    char* File1Data = 0;
    size1 = filesize(file1.c_str());
    File1Data = new char[size1+1];
    inFile.read(File1Data, size1);
    File1Data[size1] = '\0' ;
    inFile.close();

   //User input with the name of the original file (targey
    inFile.open(file2.c_str(), ios::in|ios::binary);
    char* File2Data = 0;
    size2 = filesize(file2.c_str());
    File2Data = new char[size2+1];
    inFile.read(File2Data, size2);
    File2Data[size2] = '\0' ;
    inFile.close();

    long minsize = size1;
    if (minsize > size2)
        minsize = size2;
```

```
long diffCount = 0; //Used to keep track of the difference between
                    //two files
for (long i = 0; i < minsize; i++)
{
    //ch1: looking at file character by character (original file)
   //ch2: looking at file character by character (target file)
    char ch1 = File1Data[i];
    char ch2 = File2Data[i];

    //If the character in file1 is not equal to the character in
    file2 → add to the diffCount variable
    if (ch1 != ch2)
        diffCount++;
}

string ReportString = "";
char numstr[1000];

ReportString.append("File 1 (Original): " + file1 + "\n");

sprintf(numstr, "%d", size1);
ReportString.append("Size: ");
ReportString.append(numstr);
ReportString.append("\n");

ReportString.append("\n");

ReportString.append("File 2 (Test): " + file2 + "\n");

sprintf(numstr, "%d", size2);
ReportString.append("Size: ");
ReportString.append(numstr);
ReportString.append("\n");

ReportString.append("\n");

//Displays the differences between the original and target file
ReportString.append("Number of Bytes Difference: ");
sprintf(numstr, "%d", diffCount);
ReportString.append(numstr);
ReportString.append("\n");

//Displays how much of the original file is corrupted
ReportString.append("Percentage of Original Corrupted: ");

double percor = 0;
if (size1 > size2)
    percor = ((size1-size2) + (float)diffCount)/size1;
else
    percor = (float)diffCount/size1;

sprintf(numstr, "%f", percor * 100);
ReportString.append(numstr);
ReportString.append(" % \n");

cout << ReportString << endl;
```

```
    //Saves the difference in a text file named “DifferenceReport”
    ofstream outfile("DifferenceReport.txt");
    outfile << ReportString << std::endl;
    outfile.close();

    cout << endl << "Press ENTER to continue...";
    cin.get();
    cin.get();

    return 0;
}
```

Works Cited

Amankwah-Amoah, Joseph. "Competing Technologies, Competing Forces: The Rise and Fall of the Floppy Disk,1971-2010." *Technological Forecasting & Social Change* (n.d.): 121-29. Print.

Casey, Eoghan. *Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet*. London: Academic, 2004. Print.

Güllüce, Yusuf Ziya, and Recep Benzer. "Hard Disk Failure and Data Recovery Methods in Computer ForensicAdli Bilişimde Hard Disk Arızaları Ve Arızalı Disklerden Veri Kurtarma Yöntemleri." *International Journal of Human Sciences* 12.1 (2015): 206-25. Web.

Hiatt, Charlotte J. *A Primer For Disaster Recovery Planning In An IT Environment*. Hershey, Pa: IGI Global, 2000. *eBook Collection (EBSCOhost)*. Web. 31 Oct. 2016.

Kent, Karen, Suzanne Chevalier, Tim Grance, and Hung Dang. "Guide to Integrating Forensic Techniques into Incident Response." N.p.: n.p., 2006. N. pag. Web.

Lee, Byunghee, Kyungho Son, Dongho Won, and Seungjoo Kim. "Secure Data Deletion for USB Flash Memory." *Journal of Information Science and Engineering* (2011): 933-52. Web.

Messier, Ric. *Operating System Forensics*. Waltham, MA: Syngress Is an Imprint of Elsevier, 2016. Print.

Moshenska, Gabriel. "The Archaeology of (flash) Memory." *Post-Medieval Archaeology* 48.1 (2014): 255-59. *Academic Search Complete*. Web. 29 Oct. 2016.

Pihlajamaa, Joonas. "Simple FAT and SD Tutorial Part 1." *Code and Life*. N.p., n.d. Web. 11 Dec. 2016.

Schneier, Bruce. "The Story Behind The Stuxnet Virus." *Forbes*. Forbes Magazine, 07 Oct. 2010. Web. 04 Apr. 2017. <https://www.forbes.com/2010/10/06/iran-nuclear-computer-technology-security-stuxnet-worm.html>.

Siewart, Sam, and Dane Nelson. "Solid State Drive Applications in Storage and Embedded." *Intel Technology Journal* 13.1 (n.d.): 29-53. Web.

Sommer, Peter. "Emerging Problems in Digital Evidence." *Criminal Justice Matters* 58.1 (2004): 24-25. Web.

"The Story Behind The Stuxnet Virus." *Forbes*. Forbes Magazine, 07 Oct. 2010. Web. 4 Apr. 2017. <https://www.forbes.com/2010/10/06/iran-nuclear-computer-technology-security-stuxnet-worm.html>.

"The Stuxnet Worm." Norton by Symantec. N.p., n.d. Web. 04 Apr. 2017. <https://us.norton.com/stuxnet>.