

Forensic Acquisition Utilities

Copyright © 2002, 2003 George M. Garner Jr.

Revised May 30, 2003

Project purpose and components:

This is a collection of utilities and libraries intended for forensic or forensic-related investigative use in a modern Microsoft Windows environment. The components in this collection are intended to permit the investigator to sterilize media for forensic duplication, discover where logical volume information is located and to collect the evidence from a running system while at the same time guaranteeing data integrity (e.g. with a cryptographic checksum) and while minimizing changes to the subject system while the duplication process is under way. A third party hardware or software write blocker should be employed in those circumstances where it is deemed necessary to guarantee that no changes occur to the subject volume prior to and after the imaging process.

A major focus of this release is to reduce the time that it takes to acquire a forensic image. Even home computer systems today are being equipped with hard drives in excess of 40 GB in size. Imaging a 40 GB hard drive using conventional methods (e.g. the standard versions of `dd`, `md5sum` and `netcat` on Linux) may take 6 hours or more, particularly if the output of `dd` is piped to `gzip` before being piped to `netcat`, as is a common practice. Six hours is a long time when responding to an incident and a significant cost factor in terms of lost productivity and labor at current hourly rates for incident response personnel.

Io-intensive applications such as `dd` and `netcat` spend a significant amount of time copying data from one buffer to another. To image a logical volume, volume data must be paged from the drives internal buffers to the file systems in-memory cache. The data then is copied from the system's cache into the application's read buffer. If a separate application write buffer is used, the data must be copied from the read buffer into the application write buffer. Then the data must be copied from the application's write buffer to the system output buffer or write cache. Next, the data must be copied from the system output buffer or write cache to the internal buffers of the output device. Finally, the output device must copy the data from its internal buffers to the output device. If `zlib` compression is used, the data must be copied an additional two times: Once into the `zlib` input buffer; and once, in compressed form, into the `zlib` output buffer. Piping information from one application to another adds an additional two copy operations for each pipe: Once on the sending side; and once on the receiving side. The time required for each copy operation is linear. In other words, the time required to image a volume increases at a constant rate each time the data is copied during the imaging process.

Cryptographic checksum generation is another factor that adds to the time required imaging a drive or logical volume. Standard open source tools such as `dd` and `Netcat` do not offer cryptographic

verification, which means that cryptographic verification must be done sequentially. Typically, `md5sum` will be run on the source drive or volume. The drive or volume then is imaged. Finally, `md5sum` or a comparable tool is run on the destination file or volume to verify that the copy process resulted in an authentic copy of the original.

Lack of multi-threading in traditional tools contributes substantially to the time required for imaging a drive or logical volume. I/O-threads spend a substantial part of their time blocking while waiting for the device to complete their requests for input or output. Even with only one processor, most modern operating systems permit an application to write to an output device (e.g. network drive or socket) at the same time that it is waiting for input from a local disk drive if the application is designed to do so. Most traditional forensic tools were not written with multi-threading in mind and do not adapt easily to multithreaded programming idioms.

The present release attempts to reduce the time required for volume or drive imaging by reducing, if not eliminating, the need for piping and by incorporating cryptographic verification into the imaging application. Standard UNIX utilities such as `dd`, `md5sum` and `Netcat` do not lend themselves easily to multithreading and this has not been attempted in this release.

What's included in this release:

Included in this release are the following modules:

1. `dd.exe`: A modified version of the popular GNU `dd` utility program
2. `md5lib.dll`: A modified version of Ulrich Drepper's MD5 checksum implementation in Windows DLL format.
3. `md5sum.exe`: A modified version of Ulrich Drepper's MD5sum utility.
4. `Volume_dump.exe`: An original utility to dump volume information
5. `wipe.exe`: An original utility to sterilize media prior to forensic duplication.
6. `zlibU.dll`: A modified version of Jean-loup Gailly and Mark Adler's `zlib` library based on [zlib-1.1.4](#).
7. `nc.exe`: A modified version of the `netcat` utility by Hobbit.
8. `getopt.dll`: An implementation of the POSIX `getopt` function in a Windows DLL format.

This software requires Microsoft Windows 5.0 (Windows 2000) or later. Versions of Microsoft Windows prior to Windows 2000 will not be supported. The software has been tested on Microsoft Windows 2000 Gold, Microsoft Windows 2000 SP3, Microsoft Windows XP SP1 and Microsoft .Net Server.

Each project in this distribution, `dd`, `getopt`, `md5lib`, `md5sum`, `netcat`, `volume_dump`, `wipe` and `zlib`, has an accompanying readme file. Do not use this software until you have read and understand the information in all of the readme files that accompany this distribution.

What's new in this release:

`dd` build 1033

- Fixed a problem whereby the output was sometimes recorded as zero bytes. This problem affected only the metadata in the log file and did not affect the validity of the image.

Build 1032:

- This build of `dd` fixes a problem whereby `dd` fails to interpret certain read errors as indicating an end-of-file condition. The original POSIX release of `dd` assumes that a read operation always returns zero when the read operation encounters an end-of-file condition. This assumption is not POSIX-ly correct when reading from a physical device (`\\.\PhysicalDriveX`): http://www.opengroup.org/onlinepubs/007904975/functions/read.html#tag_03_594_03 ("No data transfer shall occur past the current end-of-file. If the starting position is at or after the end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent `read()` requests is implementation-defined.") Current builds of Microsoft Windows 2000 and Windows XP return one of two error codes when a read operation on a physical drive attempts to read beyond the end of the device. The same error codes are returned when a bad sector is encountered.

In practical terms this means that in POSIX semantics the only difference between a "bad sector" and an end-of-file condition when reading from a physical drive is the location (offset) at which the error occurs. If the error occurs before the expected end of the drive then it is a "bad sector." If the error occurs at or after the expected end of the drive then it is an end-of-file condition.

In previous versions, `dd` estimated the "total size" of a drive using a formula that underestimates the actual number of addressable bytes for most modern drives:

$$\text{Size} = \text{Bytes_per_sector} \times \text{Sectors_per_track} \times \text{Tracks_per_cylinder} \times \text{Cylinders}.$$

In previous versions, the drive size was reported for informational purposes only and was not used in the imaging algorithm. In the present release, the "total size" is used when imaging a physical drive to distinguish between read errors that indicate a "bad sector" and read errors that indicate an end-of-file condition. `dd` determines the "total size" of a drive by reading the drive capacity from the drive bios.

On Microsoft Windows XP and later, reading the capacity of a drive requires only that the drive be opened for read access. On Microsoft Windows 2000, reading drive capacity requires that the drive be opened for both read and write access. While opening a drive for write access permits an application to send write requests to a drive, it does not require the application to send any write requests. Forensic investigators who use `dd` to image a drive seized from a previously halted system (traditional forensic imaging) always should use write blocking hardware or software. Forensic investigators who use `dd` to image a running ("live") system are not able to use write blocking hardware or software but may use a bus monitor (e.g. <http://www.bustrace.com>) for test purposes `ms3` to verify that `dd` does not send any write commands directly to the drive.

If logging is enabled (`--log` option), `dd` records the capacity of the drive as the "Total size" of the drive in kibibytes `ms3`. A forensic investigator should compare the capacity of a drive as reported by `dd` with the manufacturer's specification for the drive to ensure that the capacity has not been

altered in the drive bios. **dd** also will record the offset at which the imaging process ends in bytes. A forensic investigator should compare this offset to the drive capacity or "total size" to verify that the entire drive was imaged. (Note that the offset is recorded in bytes and may be converted to kibibytes by dividing by 1024.)

No change was required for imaging a logical volume ([\.\C:](#)) which continues to return zero when an end-of-file condition is reached.

- In previous versions, **dd** always recorded the number of bytes copied in both compressed and uncompressed form even if the destination file system did not support file level compression (e.g. fat32). Build 1032 of **dd** only reports the compressed value when imaging to a file system that supports file level compression.

- The input buffer now is sector aligned (aligned on addresses in memory that are integer multiples of the volume's sector size). The original POSIX release of **dd** incremented the pointer to the input buffer by two bytes to support swabbing (swapping of every pair of input bytes). This two-byte swab offset throws the input buffer out of sector alignment. Sector alignment is not enforced by many device drivers but is enforced by certain firewire/ATA bridge products that are of interest to forensic investigators. The Forensic Acquisition Utilities release of **dd** does not support swabbing. The swab offset has been eliminated from the current release.

Netcat Build 1032:

- The "Total size" of a physical drive is determined by reading the drive capacity. Note that **Netcat** does not attempt to handle "bad sectors" and did not require modification due to the end-of-file returning error when reading from physical device problem discussed above.

Md5sum Build 1032:

- Fixes an access violation that occurred when **md5sum** was run with the `-c` checkfile option but no check file was specified.

Build 1030:

1. Fixes hang in **dd** and **netcat** while reading partition table on certain Windows 2000 machines.
2. **MSVCP70.DLL** is included among the Microsoft redistributable files under the system directory and may be required to run the software.

Program Binaries:

Both debug and release binaries accompany this release. The debug binaries and associated **pdb** files may be found in the **bin/UnicodeDebug** directory. The release binaries are to be found in the **bin/UnicodeRelease** directory.

Source:

A number of the programs included in this release (`dd` , `md5lib`, `md5sum`, `getopt`) are based on programs that are part of the UnxUtils distribution available at <http://unxutils.sourceforge.net/> . The original (unmodified) source for these programs may be obtained via anonymous CVS by following the instructions on this site.

Source code for the binaries included in the forensic acquisition utilities package accompanies this release. This release targets a modern Microsoft Windows[™] environment. The requirements for building the source code into a binary executable are stated below under [Build Instructions](#).

License:

The source is available under GNU public [license](#). A copy of the license may be found accompanying this distribution.

Downloading the Forensic Acquisition Utilities:

The current release Microsoft Windows binaries of the Forensic Acquisition Utilities is build 1.0.0.1032 (beta1), which may be downloaded as a compressed zip file from [here](#). Both debug and release binaries are included. A detached PGP signature of the compressed zipped binaries is available from [here](#). The source code for the current release of the Forensic Acquisition Utilities may be downloaded from [here](#). A detached PGP signature of the compressed zipped source code is available from [here](#). The binary files that have changed since build 1030 may be downloaded as a [patch](#). The patch includes `dd` build 1033. (Note that Forensic Acquisition Utilities is build 1.0.0.1032 (beta1) does not include `dd` build 1033.) The PGP detached signature for the patch may be downloaded from [here](#).

Build Instructions:

Equipment (in order of installation):

1. A personal computer
2. Microsoft Windows 2000 or Windows XP + latest service packs and hot fixes
3. Microsoft Visual Studio .Net (with Microsoft SDK November 2001)
4. The Source Code accompanying this release

Instructions:

1. Install Microsoft Windows 2000 or Windows XP on the computer development platform.
2. Install appropriate service packs and hot fixes.
3. Install Visual Studio .Net(with Microsoft SDK November 2001)
4. Install the source code accompanying this release into project directory.
5. Open forensic acquisition utilities.sln into Microsoft Visual C++ 7.0 (a part of Visual Studio .Net).
6. On the Build menu select "Rebuild Solution."

7. Wait until build completes.

Catalog:

Catalogs, in Microsoft catalog format, which contain sha1 hashes of the [source](#) and [binary](#) files are included in this release. The catalogs were made using `makecat.exe`, which is part of the Microsoft SDK. Currently there is no way for third party ISV's to verify files based on the MS catalog format. However, I expect this functionality to become available in the next release of CAPICOM.

Microsoft CRT version 7.0:

The debug build of this release requires `MSVCR70D.DLL`, `MSVCR70D.PDB`, `MSVCP70D.DLL` and `MSVCP70D.PDB`. These files are not redistributable and are not included in this distribution but may be obtained by purchasing Microsoft Visual Studio .Net. The release build requires `MSVCR70.DLL` and `MSVCP70.DLL`. Redistributable versions of `MSVCR70.DLL` and `MSVCP70.DLL` are included with this distribution under the System directory. `MSVCR70.DLL` and `MSVCP70.DLL` also are included in Microsoft Windows XP and .Net Server but not Microsoft Windows 2000. If you are using Microsoft Windows XP or .Net Server, you may use the version of `MSVCR70.DLL` and `MSVCP70.DLL` that comes with the operating system. If you are using Windows 2000, copy `MSVCR70.DLL` and `MSVCP70.DLL` into the same directory where you install the release binaries.

Operating System Requirements:

This software requires Microsoft Windows 2000, Windows XP or .Net Server. Some functionality may not be available on Microsoft Windows 2000. See the accompanying readme files for further details. If you are working on a Linux-only platform you may want to consider using the F.I.R.E. distribution of `dd`, which incorporates md5 checksums into the imaging process. <http://fire.dmzs.com/>.

Remarks:

Incorporation of md5 checksums into `dd` required a change in the original `md5sum` digest format. Forensic specialists use `dd` to make a bit-wise image of an original data source. Md5 checksums are used to ensure that an image is a true copy of the original. The traditional GNU `md5sum` digest format permits only a single path in addition to the "stringified" md5 checksum of a file. This would allow us to record either the source path or the destination path of a bit-wise image file, but not both. If the `dd` program were to write the input path to the digest, the original `md5sum` program would require access to the original source path to verify the checksum. If the `dd` program were to write the output path to the digest, the `md5sum` program would be able to verify the image file without access to the original source, however, the digest would not contain any indication of where the image came from. The new digest format writes both the input and output paths to the digest. The input path is placed within square

brackets ("[]"). The `md5sum` program provided with this distribution verifies the digest checksum against the output path. The input path (i.e. anything within square brackets) is ignored for verification purposes. The `dd` program writes the md5 checksum of the original input to the digest, not the checksum of the output file. The checksums will not match unless the output image file is a true copy of the original. Note that the `md5lib` checksum algorithm requires that the block size be a multiple of 64. A block size equal to the page size (4KiB) is preferred and is what has been tested. Microsoft Windows XP and Windows 2000 may use large (4MiB) pages (on systems with more than 256MiB of memory).

This software has not been tested on systems using large (4MiB) page sizes.

Analysis of resultant volume image files:

Low level analysis may be done directly on the volume image files using your favorite tools. Win32 ports of many standard UNIX tools are available at <http://unxutils.sourceforge.net/>. The UnxUtils package does not include a strings utility. However, a strings utility that supports Unicode may be found at <http://www.sysinternals.com/>. On Microsoft Windows systems, `Restorer2000` may be used to analyze the file system of the volume image if the original volume was formatted to the NTFS files system. A virtual file disk driver named `filedisk` is available at <http://www.acc.umu.se/~bosse/>. An Ext2/Ext3 file system driver for Microsoft Windows NT/2000/XP may be found at http://www.partition-manager.com/n_ext2fs_main.htm. The promotional version of this driver mounts Ext2/Ext3 volumes in read only mode. On Linux systems, the `Sleuth Kit` and the `Autopsy` may be used to examine volume images.

Choice of image acquisition method and tools:

When you arrive on the scene of a probable incident you will encounter one of two possible scenarios: Either the subject system(s) is powered down; or, more probably, the system(s) is still running. If the system is powered down, there is very little debate over what to do next: Don't start it up. Rather, boot to a reliable operating system `Linux` and image the drive(s) or volume(s) using your favorite tools. `Dd` running on Linux or one of the commercial products such as `Encase` appear to be the tools of choice at the moment. The Forensic Acquisition Utilities are not intended for this scenario unless write blocking hardware or software is employed.

By far the more common scenario for incident response personnel is to find the suspect system(s) running when you arrive on the scene. What to do in this scenario has been the subject of much discussion and you will find a broad range of opinions expressed in the literature. For example, Robert E. Greenfield recommends shutting the system down "following the normal Windows shutdown process" to avoid damaging the hard drive. Robert E. Greenfield, *The Liturgical Forensic Examination: Tracing Activity on a Windows-Based Desktop* in Albert J. Marcella and Robert S. Greenfield, editors, *Cyber Forensics. A Field Manual for Collecting, Examining, and Preserving Evidence of Computer Crimes* (New York, 2002), 74.

On the other hand, it is known to be the policy of a certain government agency to pull the plug (literally) shortly after arriving on the scene. See Warren G. Kruze II and Jay G. Heiser, *Computer Forensics. Incident Response Essentials* (New York, 2002), 5 ("Deciding whether to let a machine continue to run, to pull the power plug from the back of the computer, or to perform the normal administrative shutdown process is one of the longest running arguments in the computer forensics field"). A broad variety of options are possible in between these two extremes. The choice of method is important because it may well determine the outcome of the investigation; and yet we are forced to choose a method before we

have acquired the evidence needed to validate the assumptions upon which our choice of method is based.

The preservation of file system data and metadata often is a primary goal of evidence gathering from computer systems for future forensic analysis. But what is a file system and where do file system data and metadata reside. One possible answer is that the file system resides on the surface of some medium, such as a hard drive platter. This is certainly true once a system has been powered down. But it is not entirely true of a running system. Windows, like most modern operating systems, uses a system cache to buffer file system reads and writes. Under some circumstances the system write cache may be disabled, but that is not usually the case. Modern hard drives also have large on-disk caches. The on-disk cache may be bypassed programmatically, but that also is not usually the case. On a running system, file system data and metadata exist at each of these levels: system cache, on-disk cache and physical medium. Each of these levels represents a particular slice in time of the file system. Information at each of these levels is evidence. When an investigator arrives on the scene of a running system, the investigator must decide which evidence will be preserved and which evidence will be altered or destroyed. If the investigator shuts the system down normally or flushes file buffers, evidence on the physical medium will be altered and the original state of the physical medium will be lost. If the investigator pulls the plug, information on the physical medium may (hopefully) be preserved, but the information in the system cache and on-disk cache will be destroyed. If the investigator tries to preserve both by imaging physical memory prior to pulling the plug, then both may end up changed.

Deciding which method is the best method in an individual case depends upon a variety factors, including one's corporate or agency policy, the nature of the case, what is alleged or suspected, what has been happening to other similar systems recently, what human and other resources are on hand to respond to the incident, business and economic considerations, and so on. The answer to the question may change over time in response to exigent circumstances; and in all probability, it would be wise to vary the methods used to slow the development of effective counter-measures [3336]. Nevertheless, the choice of method will dictate the appropriate choice of tools and whether the tools in this package are right for you. The tools in this package are intended for use on a running system. These tools are not intended for use once the power has been disconnected (for whatever reason) from the subject system unless you have access to a hardware or software write-blocker. If the subject system is running when you arrive on the scene and you decide (for whatever reason) to gather evidence from the running system, then you may find the tools included in this package useful. It is hoped that they will increase the range of options available to incident response personnel and other investigators.

When collecting evidence from a "live" system to a network share on a forensic workstation running Microsoft Windows 2000 or XP, first disable file caching on the network share of the forensic workstation. Otherwise, the evidence that has been written to the network share may be cached back on the subject system just as if it were written there in the first place.

The versions of `Dd` and `Netcat` included in this release compute a cryptographic checksum (MD5 hash) of data "on-the-fly," i.e. as the data is collected. The cryptographic checksum is intended to guarantee the integrity of the evidence once the evidence has been read into `DD` or `NETCAT` application buffers. The checksum is not intended to guarantee the integrity of core system binaries upon which the read operation depends (e.g. file system, storage and bus drivers and any attached upper or lower filter drivers). Methods for validating core system binaries on a "live" system are not addressed in this release.

Automation of the evidence collection process is highly desirable, particularly when evidence is being collected from a “live” system. The versions of **Dd** and **Netcat** provided in this release support a “—log” option that automates to a certain extent the logging of the circumstances and process of evidence collection by these tools. Aside from this automated logging feature, this release does not provide any further means of automation. The [Forensic Server Project](#) is intended to provide a degree of automation in the collection of volatile data from “live” systems. The project does not address the acquisition of a forensic image from a live system. Nevertheless, administrators and incident response personnel may find this useful in collecting other volatile data.

Examples:

```
dd.exe if=\\.\PhysicalDrive0 of=d:\images\PhysicalDrive0.img --md5sum --verifymd5 --md5out=d:\images\PhysicalDrive0.img.md5
```

```
dd if=\\?\Volume{87c34910-d826-11d4-987c-00a0b6741049} of=d:\images\e_drive.img --md5sum --verifymd5 md5out=d:\images\PhysicalDrive0.img.md5
```

```
dd.exe if=\\.\PhysicalMemory of=d:\images\PhysicalMemory.img bs=4096 --md5sum --verifymd5 --md5out=d:\images\PhysicalMemory.img.md5
```

```
dd.exe if=\\.\D: of=d:\images\d_drive.img conv=noerror --sparse --md5sum --verifymd5 --md5out=d:\images\d_drive.img.md5 --log=d:\images\d_drive.log
```

```
dd.exe if=myfile.txt.gz of=d:\images\myfile.txt conv=noerror,decomp --md5sum --verifymd5 --md5out=d:\images\myfile.txt.img.md5 --log=d:\images\myfile.txt.log
```

```
dd.exe if=\\.\D: of=d:\images\d_drive.img.gz conv=noerror,comp --md5sum --verifymd5 --md5out=d:\images\d_drive.img.md5 --log=d:\images\d_drive.log
```

```
md5sum.exe -o d_drive.md5 \\.\D:  
md5sum.exe -c d_drive.img.md5  
md5sum.exe -d zlib -c d_drive.img.gz.md5
```

```
nc -v -n -l -p 3333 -csum md5 --verify --sparse -O myimage.img.  
nc -v -n -csum md5 -l \\.\C: 192.168.0.1 3333
```

```
nc -v -n -l -p 3333 --comp zlib -O myimage.img.gz.  
nc -v -n -l \\.\C: 192.168.0.1 3333
```

```
nc -v -n -l -p 3333 -csum md5 --verify -O myimage.img.gz.  
nc -v -n -lock -csum md5 --comp zlib -l \\.\D: 192.168.0.1 3333
```

```
nc -v -n -l -p 3333 -csum md5 --verify --sparse -O myimage.img.  
nc -v -n -lock -csum md5 -l \\?\Volume{87c34910-d826-11d4-987c-00a0b6741049 } 192.168.0.1 3333
```

volume_dump

```
volume_dump \\.\C:
```

```
wipe \\.\D:
```

```
wipe c:\temp\myfile.txt
```

```
wipe c:\temp\myfile.*
```

```
wipe c:\temp\myfil?.txt
```

```
wipe \\.\Tape0
```

```
wipe -p 1 \\.\Tape0
```

Windows XP only:

The WebDAV redirector may be used to write images and other data to a WebDAV enabled web server. To use the WebDAV redirector, first map the WebDAV site as a logical drive using the net use command (e.g. "net use X: http://mywebserver/images"). Then specify the mapped drive as the output directory on the command line. This has been tested without authentication using IIS and Apache WebDAV servers. In a real world scenario, authentication would be used to control access to the WebDAV server. Authentication is transparent with IIS. Basic authentication has been tested and does not work correctly with Redhat 7.2 due to problems with the format of the user name sent from the XP client (server \username or username@mydomain.com). Other forms of authentication (e.g. certificate based authentication) have not been tested. Dd over SMB (CIFS) file shares and netcat generally provide better performance than dd over WebDAV. On the other hand, WebDAV is a protocol that is supported on both Windows XP and Linux platforms and therefore provides an additional option for cross-platform imaging.

Known Bugs:

Forensic Acquisition Utilities-1.0.0.1032 (beta1)

1. **Netcat** is able to write a compressed stream to the net. If the compressed stream is subsequently written to a file in compressed form by **netcat**, acting in listening mode, verification of the stream (--verify option) will erroneously fail. This is because the transmitted hash was made from the original decompressed data and the **netcat** listener does not know that it needs to perform checksum verification on the decompressed stream. To verify compressed data received by the **netcat** listener, use the **md5sum** release that comes with this solution ("md5sum.exe -d zlib -c myfile.gz.md5"). This problem will be fixed in a future release.

2. Access to raw physical drives and logical volumes requires administrative privileges on Microsoft Windows 2000 and Windows XP. If you attempt to copy a physical drive or logical volume using **dd** and are not logged on as an administrator, **dd** will present you with the secondary logon dialog. If you enter the appropriate name and password, the secondary logon will succeed. However, **dd** subsequently will fail if the **-log** option has been specified on the command line. This is because the log file is opened read only and the spawned administrative process is unable to open it. This problem will be fixed in a subsequent release.

3. This release of **Netcat** does not provide any means of writing log output to the net as opposed to the console of the subject system (--log option). This will be fixed in a future release. In the meantime, log output may be "piped" to a second copy of **Netcat**.
4. Except for **Wipe**, writing to a tape drive (e.g. \\.\Tape0) is not supported. Adding tape support will be considered for a future release.
5. Application manifests may be used on Microsoft Windows XP to ensure the integrity of dependent assemblies but are not included in this beta distribution. The final release will include application manifests for release binaries.
6. If a file is read-only, **Wipe** will fail to wipe the file but the error message does not indicate that the reason for the failure is because the file is read-only. This will be fixed in a future release either by overriding the read-only attribute or by displaying a more informative error message. Comments would be welcome as to which approach should be taken.
7. If **dd** is invoked to copy a regular file and no incompatible options are specified (e.g. compression, sparse files, checksums, block sizes), **dd** will copy the file using the Win32 [CopyFileEx](#) function. When [CopyFileEx](#) is used to copy a regular file from a NTFS volume to another NTFS volume, [CopyFileEx](#) will copy alternate streams in addition to the primary stream. With this one exception, **dd** and **netcat** do not copy alternate streams. This functionality may be considered for a future release depending upon user feedback.
8. The physical memory "imaged" by **DD** and **NETCAT** is allocated in units of 4KiB or 4MiB pages. There is no inherent relationship between the logical order of pages and their physical order so that the contents of one physical page need not be related to the physical pages adjacent to it. One needs to know the page size effective on a system (4KiB or 4MiB) to properly interpret the physical memory "image" taken from the system. **NETCAT** and **DD** do not properly record the page size in effect. This problem will be fixed in a future release.
9. **Dd** and **Netcat** will write to a local drive or file if a local drive or file is specified as the output file on the command line. This is by design since these tools may be used not only to collect evidence but also to duplicate evidence or to restore evidence (e.g. by imaging a drive) for purposes of analysis. However, evidence also may be destroyed by inadvertently specifying a local drive or file as the output file. At present there is no warning that this is about to happen. This problem will be fixed in a future release.
10. **Dd** will seek past sectors that cannot be read ("bad sectors") if the noerror option ("conv=noerror") is specified on the command line. This permits an image to be obtained even from damaged media or media with "bad sectors." However, **dd** increments the file pointer in increments equal to the block size. Data may be lost if **dd** is used to image a physical device and the block size is not equal to the logical sector size of the drive. The default block size for this release of **dd** is 4096 bytes which provides optimum performance on Microsoft Windows. This should not present a problem in most cases when imaging most modern hard drives. Most modern hard drives hide "bad sectors" from the operating system using a process known as sparing. The log will indicate if "bad sectors" were encountered during the imaging process and the offset at which they were encountered. When imaging other media such as

floppy disks, the block size should be set equal to the logical sector size of the media using the "bs" command line option (e.g. "bs=512"). Note that this limitation also occurs in the original POSIX release of `dd`.

11. The Forensic Acquisition Utilities do not conform to current NIST standards for binary and decimal prefixes. <http://physics.nist.gov/cuu/Units/prefixes.html>; and <http://physics.nist.gov/cuu/Units/binary.html>. In the current release, decimal prefixes (K, M, G) are used erroneously in place of binary prefixes (Ki, Mi, Gi). This problem will be fixed in a future release.

Bugs:

Report bugs to gmgarner@erols.com.

Acknowledgements:

Thanks to [Rob Lee](#) for reference to [Restorer2000](#) and for his help in testing this software prior to the release of this beta. Thanks to Paul Livingston for reviewing the main readme file prior to release. Thanks to Mark Scott for reporting the problem with `dd` failing to handle correctly an end-of-file condition when reading from a physical drive.