COMPRESSING AND ARCHIVING

WHAT IS COMPRESSION?

Compression, as the name implies, makes data smaller, thereby requiring less storage capacity and making the data easier to transmit. For your purposes as a beginning hacker, it will suffice to categorize compression as either lossy or lossless. Lossy compression is very effective in reducing the size of files, but the integrity of the information is lost. In other words, the file after compression is not exactly the same as the original. This type of compression works great for graphics, video, and audio files, where a small difference in the file is hardly noticeable—.mp3, .mp4, .png, and .jpg are all lossy compression algorithms. If a pixel in a .png file or a single note in an .mp3 file is changed, your eye or ear is unlikely to notice the difference—though, of course, music aficionados will say that they can definitely tell the difference between an .mp3 and an uncompressed .flac file. The strengths of lossy compression are its efficiency and effectiveness. The compression ratio is very high, meaning that the resulting file is significantly smaller than the original. However, lossy compression is unacceptable when you're sending files or software and data integrity is crucial. For example, if you are sending a script or document, the integrity of the original file must be retained when it is decompressed. Unfortunately, lossless compression is not as efficient as lossy compression, as you might imagine, but for the hacker, integrity is often far more important than compression ratio.

TARRING FILES TOGETHER:

Usually, the first thing you do when compressing files is to combine them into an archive. In most cases, when archiving files, you'll use the tar command. Tar stands for tape archive, a reference to the prehistoric days of computing when systems used tape to store data. The tar command creates a single file from many files, which is then referred to as an archive, tar file, or tarball.

```
Syntax> tar -cvf   tarfile.tar  file1 file2 file3 …so on
```

```
Kali> tar -cvf  file.tar  file1 file2
```

Where,

```
-c          create
```

```
-v          details or verbose
```

```
-f          write to the following file
```

We can display those files from the tarball, without extracting them, by using the tar command with the -t content list switch, as shown next:

```
Kai> tar  -tvf file.tar
```

To extract we can use the following command:

```
Kali> tar  -xvf  file.tar
```

COMPRESSING FILES:

Now we have one archived file, but that file is bigger than the sum of the original files. What if you want to compress those files for ease of transport? Linux has several commands capable of creating compressed files. We will look at these:

gzip, which uses the extension .tar.gz or .tgz ➔ in between bzip2 and compress

bzip2, which uses the extension .tar.bz2 ➔ slowest but smallest resultant files

compress, which uses the extension .tar.z ➔ fastest but larger resultant files

These all are capable of compressing our files, but they use different compression algorithms and have different compression ratios. In general, compress is the fastest, but the resultant files are larger; bzip2is the slowest, but the resultant files are the smallest; and gzip falls somewhere in between.

COMPRESSING WITH GZIP:

```
Kali> gzip file.tar
```

When we do a long listing on the directory, we can see that file.tar has been replaced by file.tar.gz.

```
Kali> gunzip  file.tar.gz
```

Once uncompressed, the file is no longer saved with the .tar.gz extension but with the .tar extension instead.

COMPRESSING WITH BZIP2:

```
Kali> bzip2 file.tar
```

```
Kali> bunzip2  file.tar.bz2
```

COMPRESSING WITH COMPRESS:

```
Kali> compress file.tar
```

```
Kali> uncompress file.tar.z
```

We can also use the gunzip command with files that have been compressed with compress.

CREATING BIT-BY-BIT OR PHYSICAL COPIES OF STORAGE DEVICES:

Within the world of information security and hacking, one Linux archiving command stands above the rest in its usefulness. The dd command makes a bit-by-bit copy of a file, a filesystem, or even an entire hard drive. This means that even deleted files are copied (yes, it's important to know that your deleted files may be recoverable), making for easy discovery and recovery. Deleted files will not be copied with most logical copying utilities, such as cp. Once a hacker has owned a target system, the dd command will allow them to copy the entire hard drive or a storage device to their system. In addition,

those people whose job it is to catch hackers—namely, forensic investigators—will likely use this command to make a physical copy of the hard drive with deleted files and other artifacts that might be useful for finding evidence against the hacker. It's critical to note that the dd command should not be used for typical day-to-day copying of files and storage devices because it is very slow; other commands do the job faster and more efficiently. It is, though, excellent when you need a copy of a storage device without the filesystem or other logical structures, such as in a forensic investigation.

```
Syntax> dd if=input_file  of=output_file
```

```
Kali> dd if=/dev/sdb  of=/root/flashcopy
```

Numerous options are available to use with the dd command, and you can do a bit of research on these, but among the most useful are the ***noerror*** option and the ***bs (block size)*** option. As the name implies, the ***noerror*** option continues to copy even if errors are encountered. The ***bs*** option allows you to determine the ***block size*** (the number of bytes read/written per block) of the data being copied. By default, it is set to 512 bytes, but it can be changed to speed up the process. Typically, this would be set to the sector size of the device, most often 4KB (4,096 bytes). With these options, your command would look like this:

```
Kali> dd if=/dev/sdb  of=/root/flashcopy  bs=4096  conv:noerror
```