Hutch Jorgensen
CS5959 – Writing Solid Code
1/28/2014

Compress Test

Compress version 4.0.0 can be found at http://ftp.sunet.se/pub/usenet/ftp.uu.net/comp.sources.unix/volume2/compress4.0/, the main file that you need is part1.gz. Open the file in an editor and as the comments say near the top delete everything above the comment "#! /bin/sh", at command prompt type "/bin/sh part1" and it will unpack the main files you need. Create a file, with no extension, called "USERMEM" and put 1000000 as text in the file. I modified the Makefile since it tries to build some other helper utilities that are in the part2 file, but I didn't use those. In the Makefile take out any reference to atob or btoa, and delete any sections that deal directly with them. In addition, delete the make install section, I could never get the install to work correctly. In the compress section add the -lm flag so that it can link correctly with some needed libraries. From there I would just put in "make compress" to build the executable. I also added flags for code coverage, but you can decide if you would like to do that also.

Compress uses an open addressing double hashing table on the prefix code / next character combination to compress a file. The double hashing means it first hashes normally, but if there are any collisions it uses a second, different hash to determine the interval for probing. To the best of my understanding, this means that it saves space by finding similar portions of the file that take up larger chunks of space, hashing that portion into a hash table, and then replacing the parts of the file with the key to the frequently used portion of data. The hash table gets embedded into the file so that it can be later used to uncompress/replace the hashed keys and rebuild the file to it's original state.

I wasn't able to directly crash the compress utility using fuzzed files. The utility simply looks through the file, finds similar chunks of characters etc. The state of the data inside of the file has no real impact on the utility. Compress is smart enough to check and recognize if it does not have any impact on the size of the file then it sees no reason to create the new file. Fuzzing has the effect of changing the characters in the file, and if enough of these characters are changed then any repetition is destroyed so the algorithm will not be able to effectively hash anything.

To give some numbers towards this fuzzing effect I wrote a tester that runs 100 compress calls on three different file sizes: about 120 bytes, 22 kb, and 4 mb, each of which normally will compress using the utility. On the smallest file when I fuzz 30 bits it defeats the utility from being able to compress about 68% of the time, on the medium file fuzzing 5,000 bits it no longer compresses about 100% of the time, and for the largest file fuzzing 1,000,000 bits prevents compression 0% of the time. Even though in each case I fuzz about a quarter of the bits in the file, there is so much more opportunity for repetition in the largest file that the fuzzing has no effect on whether the utility will find something to compress.

Running just these basic fuzzing tests leads to about 50% code coverage as shown below by the coverage utility output:

```
gcov -b compress
File 'compress.c'
Lines executed:47.89% of 426
```

Branches executed:58.02% of 262
Taken at least once:40.84% of 262
Calls executed:27.00% of 100

The utility also implements the decompression method used for uncompress and there are many Debugging precompile sections that I didn't run and I believe this would account for the majority of what wasn't covered.

This utility wasn't replaced for any particular instability in it's code, but because there were patents surrounding the algorithm it uses, so no free version was allowed to be used at the time. GZIP was developed as a free alternative that could be included in the GNU project, and as a bonus it has better compression ratios anyway. The patent expired in 2006 so I believe this is why it is available for use.