Josh Kunz
xargs Analysis
January 30, 2014

## 1 Overview

`xargs` is a standard UNIX utility (from the findutils utility package) for converting file contents into arguments for other programs. Typically `xargs` is employed when the user wishes to run a command over a set of input files. For example, if a user had a directory full of `*.wav` files that they wanted to convert to `*.mp3` files, they might use xargs like so:

```
find . -name "*.wav" -print0 | xargs -0 -n1 lame -V0
```

The `find` utility is used to build a list of `wav` files which is then passed to `xargs`. `xargs` converts this input to a list of arguments (in this case each files is an argument) and then passes those arguments to utility program, in this case a popular wav to mp3 conversion tool `lame`. Here we use `xarg`'s `-n` argument to ensure that only one file is passed to `lame` at a time. Without this option `xargs` will try to pass as many arguments to the utility program as it can.

From this command we can already see some peculiarities of the `xargs` command, for example `xargs`'s `-0` option. Under normal operation, input to `xargs` is *parsed* according to the conventions normally used by shells. Input lines are split into arguments on whitespace (so the line `abc def` would be parsed a two inputs `abc` and `def`), whitespace can be escaped with quotes (`"abc def"` becomes one argument) and quote characters can be escaped with backslashes (`"abc \" def"` is parsed as `abc " def`).

What this means, is that if you want run a command over a set of files, those files have to be properly escaped before they are given as input to `xargs` . This can be very confusing for users of `xargs` who might expect it to simply use each line as argument instead of parsing each line into arguments. This confusing behavior should be considered before `xargs` is relied upon.

As can be seen above `xargs` has added a work around for this problem, the `-0` option. This option causes `xargs` to treat everything between two null-terminators (or a null-terminator and an `EOF`) as an argument, no parsing is done on the input.

## 2 Analysis

The lab currently has two versions of `xargs` installed 4.1, the oldest release still available from the findutils project. It was released in 1994. The lab *also* has version 4.4.2 installed the newest version of the `xargs` utility. However, the default version is version 4.1. Since this version is severely outdated, and a newer version has already been installed, version 4.4.2 should be configured as the default as soon as possible. I was not able to find any bugs that directly affected security, but this very outdated version does have a number of logic bugs in it.

> *Note: version 4.1 was in-fact so old that I was unable to compile it on any relatively 'modern' system I control. The analysis presented below was done on version 4.2.25, the next release version.*

`xargs` itself already comes with a fairly comprehensive set of tests built on the DejaGnu test system. I compiled `xargs` with the `clang` compiler's undefined behavior sanitizer, and ran the test-suite. To my surprise, `xargs` didn't invoke *any* undefined behavior. I also compiled `xargs` using `gcc`'s code coverage tools, and then ran the test-suite. `xargs`'s supplied test-suite achieved 73% code coverage. Most un-covered code was system-call error handling code, and code whose reachability is platform dependent (though most of that code was quite simple). I attempted to test system-call error handling with the Murphy fuzzing tool, but was unable to get the tool working within the alloted time.

I also tested `xargs` for memory related bugs by running it under valgrind. I supplied a several gigabyte input file and detected no memory leaks. In fact, I noticed that `xargs` allocates the same amount of memory for every run, regardless of input size. From inspection of the code, it looks like this is because `xargs` enforces hard limits on the size (and number) of arguments passed to its child program.

One last note, `xargs` has built in support for concurrent execution. If the `-P` flag is passed to the application, `xargs` will run multiple instances of the utility application. `xargs` uses the traditional UNIX fork/exec approach to concurrency so the main vector for race conditions will be through the filesystem. If two applications access the same file, race conditions may be triggered, however this is a problem inherent in concurrently running arbitrary utility programs. If synchronization is needed, a custom tool should probably be used instead.

From my cursory inspection of the `xargs` utility, it appears to be a well tested, and robust (though not stylistically well-written) piece of code. It's unlikely that `xargs` cause any issues, yet some errors may results from the confusing way in which `xargs` handles input.